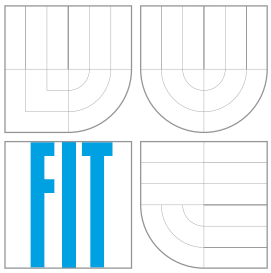
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOMPRESE OBRAZU POMOCÍ VLNKOVÉ TRANSFORMACE

IMAGE COMPRESSION USING THE WAVELET TRANSFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ PŘIKRYL
AUTHOR

VEDOUCÍ PRÁCE — Ing. DAVID BAŘINA
SUPERVISOR

BRNO 2013

Abstrakt

Práce se zabývá kompresí obrazu pomocí vlnkové transformace. Konkrétně jsou k tomuto účelu použity algoritmy EZW, SPIHT a EBCOT Tier1. Tyto algoritmy jsou pak srovnány se standardem JPEG a JPEG2000. Dále jsou zde, ke srovnání vlivu kvality komprese obrazu, použity dlaždice, různé typy vlnek a nebo barevný model RGB a $Y'C_bC_r$.

Abstract

The thesis discusses the image compression using the wavelet transform. The compression itself is done by EZW, SPIHT or EBCOT Tier1 algorithms. These algorithms are then compared to JPEG and JPEG2000 standards. Furthermore tiles, various wavelets and RGB and $Y'C_bC_r$ color spaces were used to determine the influence on an image compression quality.

Klíčová slova

vlnková transformace, vlnka, komprese obrazu, kódování, ezw, spiht, ebcot, dlaždice, RGB, $Y'C_bC_r$

Keywords

wavelet transform, wavelet, image compression, coding, ezw, spiht, ebcot, tiles, RGB, $Y'C_bC_r$

Citace

Lukáš Přikryl: Komprese obrazu pomocí vlnkové transformace, bakalářská práce, Brno, FIT VUT v Brně, 2013

Komprese obrazu pomocí vlnkové transformace

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Davida Bařiny. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Přikryl
15. května 2013

Poděkování

Děkuji mému vedoucímu panu Ing. Davidu Bařinovi za jeho odbornou pomoc při vypracovávání této práce.

Obsah

Kapitola 1

Úvod

V posledních dvou dekádách se začalo v obrazovém průmyslu diskutovat o kvalitě ztrátové komprese a jejího zlepšení. Jedna z možností byla využití vlnkové transformace a jejího nevyužitého potenciálu. Problém ovšem byl ve výpočetní náročnosti této metody. I přesto, že se na světě objevila její nejrychlejší implementace, tzv. lifting verze [7], se transformace prováděla, i na relativně malých datech, nepřiměřeně dlouho. To ovšem lidi neodradilo, a dali vzniknout mnoho algoritmů, využívající tuto transformaci, včetně algoritmů EZW (Embedded Zerotree Wavelet), anebo jeho přímého následníka SPIHT (Set Partitioning In Hierarchical Trees).

S přibývajícím výkonem počítačů již dnes není výkonostní problém takovou překážkou a na svět přichází, zřejmě nejlepší standard pro kompresi obrazu, JPEG2000. My si tento nejnovější standard otestujeme a porovnáme s již zmíněnými, staršími, algoritmy.

Nejprve si ovšem musíme definovat vlnky a jejich vznik. Teprve poté můžeme navázat na její transformaci. Zjistíme, že jich existuje více druhů. Protože ve výpočetní technice pracujeme pouze s diskrétními daty, bude nás především zajímat její diskrétní vyjádření.

Dále se podíváme na implementační záležitosti, kde si vysvětlíme nutné kroky k úspěšnému dokončení vybraných algoritmů, včetně algoritmu EBCOT (Embedded Block Coding with Optimal Truncation) a její první úrovně Tier1.

Nakonec budou otestovány některé způsoby úpravy obrazu, které mohou ovlivnit kvalitu komprese, jako jsou dlaždice, použitý barevný model, anebo použití různých vlnek. Vznikne nám tak sada testů, které budou znázorněny graficky nad různými druhy obrazů, včetně umělých geometrických tvarů a nebo také fotografií.

Kapitola 2

Teorie

Tato kapitola popisuje nutné postupy a metody zpracování obrazových dat, které jsou potřeba znát ještě před samotnou implementací. Nejprve je popsáno předzpracování obrazu, kde je užitečné vědět, co je potřeba udělat před samotnou transformací, tzn. použít správný barevný model, zvolit velikost dlaždice a nebo vybrat vhodnou vlnku.

Dále je popsán postup vzniku vlnek a vlnkové transformace a využití jejich transformačních koeficientů ke kompresi obrazu daným algoritmem.

Jsou zde popsány 3 algoritmy. EZW, na který se dále aplikuje aritmetický kodér. SPIHT, který vychází z EZW a značně ho vylepšuje. EBCOT, který je používán standardem JPEG2000.

2.1 Předzpracování obrazu

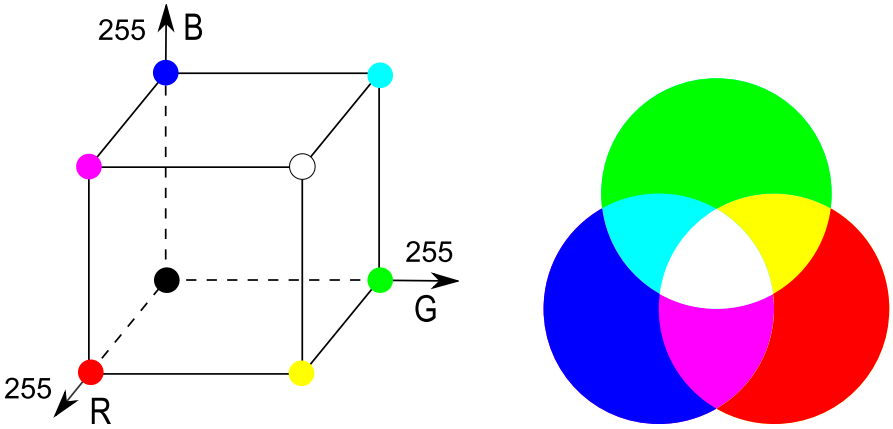
Nejprve si popíšeme, které úkony se často provádí před samotnou kompresí obrazu. Tyto úkony nám do jisté míry zlepšují kvalitu komprese, anebo vylepšují efektivitu zpracování výpočtů jejich zparalelizováním.

2.1.1 Barevné modely

Existuje několik druhů barevných modelů. Každý se hodí k jiným účelům, např. CMYK je používán v tiskárnách, RGB v zobrazovací technice (monitory, televize) a $Y'C_bC_r$ je vhodný pro ztrátovou kompresi.

RGB

RGB je vícesložkový model, tzn. k vytvoření obrazu se používá míchání více barev. Tento model se skládá ze 3 barevných složek: červená (R), zelená (G) a modrá (B). Ve 24 bitovém obraze mají tyto složky 8 bitů každá, tzn. každá RGB složka má 256 odstínů, kde 0 je nejmenší intenzita a 255 je nejvyšší intenzita. Jejich míchání je zobrazeno na obr. 2.2.

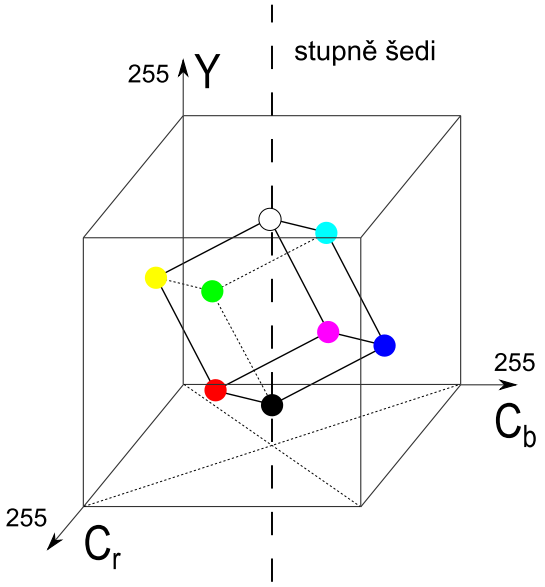


Obrázek 2.1: RGB kostka a míchání složek při jejich maximálních intenzitách

$\mathbf{Y'C_bC_r}$

$Y'C_bC_r$ je také více složkový model, ovšem k zobrazení určité barvy používá jiné barevné složky: odstín šedi neboli jas (Y), chromatická modrá (C_b) a chromatická červená (C_r).

Protože je lidský zrak více citlivý na jas nežli na kontrast barev, lze při určité ztrátě informací v chromatických složkách docílit lepší kvality obrazu při stejné kompresi než u RGB modelu s tím rozdílem, že by se chromatické složky komprimovaly více než jasové.



Obrázek 2.2: Barevný model $Y'C_bC_r$ vyjádřen pomocí natočení RGB kostky

Konverze RGB na $\mathbf{Y'C_bC_r}$ dle JPEG:

$$\begin{array}{lcrrrr} Y & = & 0{,}299R & +0{,}587G & +0{,}114B & \\ C_b & = & -0{,}168736R & -0{,}331264G & +0{,}5B & +128 \\ C_r & = & 0{,}5R & -0{,}418688G & -0{,}081312B & +128 \end{array} \tag{2.1}$$

A přepočet zpět na RGB:

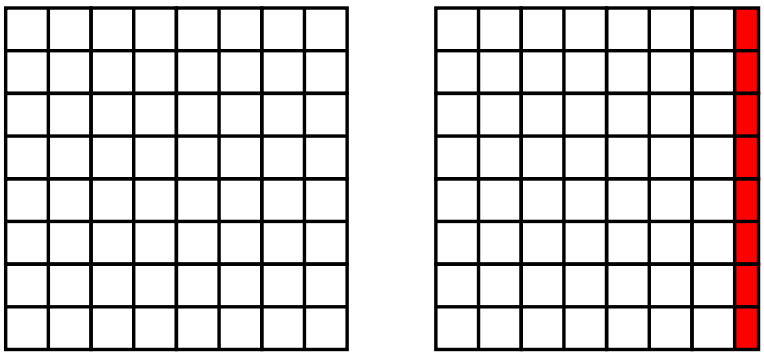
$$\begin{array}{rcrrr} R & = & Y & & +1{,}402(C_r - 128) \\ G & = & Y & -0{,}34414(C_b - 128) & -0{,}71414(C_r - 128) \\ B & = & Y & +1{,}772(C_b - 128) & \end{array} \tag{2.2}$$

2.1.2 Rozdělení obrazu na dlaždice

Tuto metodu používá velice znám a rozšířen standard JPEG, kde tímto způsobem eliminuje nevýhody diskrétní kosinové transformace. JPEG používá dlaždice o velikosti 8×8 pixelů a při vyšší úrovni komprese se v obraze objevují tzv. artefakty, způsobené právě touto metodou. Jedna z výhod je ovšem fakt, že rozdělení obrazu na menší části je méně náročné na paměťové a výpočetní zdroje.

U vlnkové transformace lze použít libovolně velké dlaždice (tak jak je tomu např. u JPEG 2000). Princip metody je zřejmý. Obraz je rozřezán na menší bloky dat (většinou stejné velikosti) a ty se následně komprimují odděleně (lze tedy např. u každého bloku zvolit jiný kompresní poměr nebo jinou vlastnost). Aplikace pak musí zajistit, aby se tyto bloky daly rozlišit od obrazových dat a mohly se tak dekomprimovat. Musí také zajistit jejich správné pozicování a zařadit je v odpovídajícím pořadí opět zpět do obrazu na své místo. Lze tak např. dekomprimovat jenom určité dlaždice a zobrazit vybraný výřez obrazu.

Rozdělení se provádí fixní velikostí. Poté se musí řešit ty bloky, které se nevejdou na okraji do obrazu. Řešení je takové, že se zvolí jako blok zbývající část. Další možností je použít velikost takovou, která by odpovídala podílu daného rozměru obrazu. Poté by se všechny dlaždice vlezly i na okraje a nemusely by se tak ořezávat. Příklad rozdělení obrazu je na obrázku 2.3.



Obrázek 2.3: Obraz rozdělený na dlaždice a vpravo je znázorněn problém „nedořezků“ na okrajích.

2.2 Vlnky

Vlnky jsou funkce, které slouží k vlnkové transformaci, tzn. transformovaný signál se porovnává s vlnkou a její deformací (roztažení a zúžení - dilatace, časový posun - translace) a jako výsledek dostaneme koeficienty, které nám říkají, jak moc je daná vlnka podobná porovnávanému signálu. Je mnoho druhů vlnek (stovky), ty jsou řazeny do určitých kategorií (rodin) podle jejich specifických vlastností a tedy i použití, např. rodina *Daubechies* má asymetrické vlastnosti, vlnky rodiny Symlet jsou téměř symetrické.

2.2.1 Vznik vlnek

Vznik vlnek sahá až do roku 1910, kdy maďarský matematik Alfréd Haar studoval ortogonální systémy a definoval funkci (2.44), dnes známou jako *vlnka Haar (Haarova vlnka)*. Tuto funkci transformoval pomocí rekurzivních součtů se vstupními hodnotami a výsledkem byla posloupnost rozdílů. V té době se o tomto postupu nehovořilo jako o vlnkové transformaci a o využití se moc nevědělo.

Později, v 70. letech téhož století, francouzský fyzik Jean Morlet při studování využití *krátkodobé Fourierovy transformace (STFT)* pro analýzu odražených signálů od vrstev horniny v geologii zjistil, že fixní délka časového okna (krátké úseky signálu) *STFT* není pro analýzu signálu tou správnou cestou k úspěchu a zvolil zcela opačný přístup. Frekvence v okně nechal konstantní a měnil pouze jeho velikost. Všiml si, že roztažením (dilatací) okna se roztahuje i funkce uvnitř.

Poté, v roce 1981 spolu s jiným francouzským fyzikem Alexem Grossmanem pracoval na aplikování jeho nového přístupu na transformaci signálu do jisté vlnové notace a následným převedením zpět na původní signál a to bez ztráty informace. Nakonec v roce 1984 úspěšně dokončili svůj výzkum a pro vlnovou funkci byl definován termín *vlnka (anglicky wavelet)* a tedy pro její transformaci pak termín *vlnková transformace (WT)*.

2.2.2 Definice vlnek

Jak již bylo nastíněno, vlnka je funkce označována jako $\psi(t)$, která musí splňovat určitá pravidla [3]. Matematicky jsou tyto pravidla definována jako

$$E = \int_{-\infty}^{\infty} |\psi(t)|^2 \, \mathrm{d}t < \infty, \tag{2.3}$$

$$C_g = \int_{0}^{\infty} \frac{|\hat{\psi}(f)|^2}{f} \, \mathrm{d}f < \infty. \tag{2.4}$$

$$\hat{\psi}(f) = \int_{-\infty}^{\infty} \psi(t) e^{-i(2\pi f)t} \, \mathrm{d}t, \tag{2.5}$$

$$\int_{-\infty}^{\infty} \psi(t) \, \mathrm{d}t = 0, \tag{2.6}$$

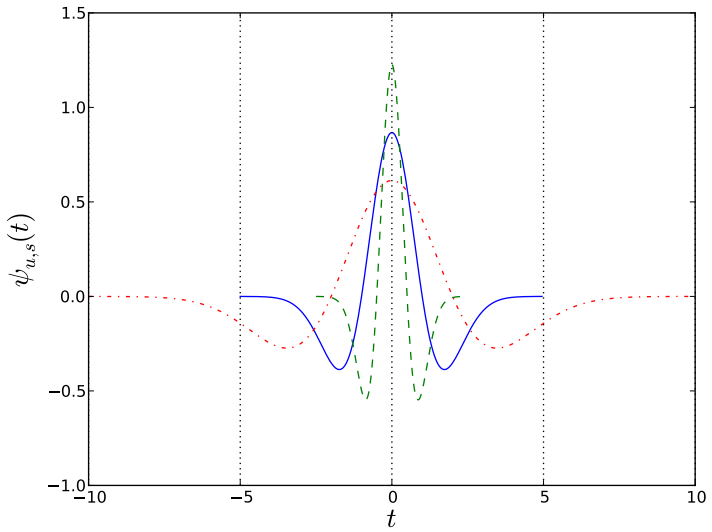
Každá vlnka musí mít konečnou energii (2.3) a nesmí obsahovat stejnosměrnou složku (2.4), tzn. její střední hodnota je nulová (2.6) a má tedy vlastnosti jako *pásmová propust*, což je také velice důležitý poznatek. Dále vlnka $\psi(f)$ má také svoji *Furierovu transformaci (FT)* $\hat{\psi}(f)$ (2.5), navíc pro komplexní vlnky musí být jejich FT reálná pro obě složky a musí zanikat pro záporné frekvence. Rovnice (2.4) se také nazývá *podmínkou přípustnosti* a C_g poté *konstanta přípustnosti*, která zaručuje invertibilitu vlnkové transformace, tj. bez ztráty informace.

2.3 Spojitá vlnková transformace

Aby bylo možné transformovat signál pomocí vlnky, je potřeba definovat operace dilatace s (změna měřítka) a translace u (časový posun) vlnky:

$$\psi_{u,s}(t) = \frac{1}{\sqrt{s}} \psi \left(\frac{t-u}{s} \right), \tag{2.7}$$

vlnka je normalizována na $\|\psi_{u,s}\| = 1$ pomocí váhové funkce $\frac{1}{\sqrt{s}}$, tzn. při změně měřítka je zachována její energie, neboli při zúžení vlnky se zvětšuje její amplituda a naopak, tak jak můžeme vidět na obrázku 2.4. Původní vlnka se také nazývá *mateřská vlnka*.



Obrázek 2.4: Normalizovaná vlnka *Mexický klobouk* pro $s = 0{,}5$, 1,0 a 2,0.

Pak si můžeme definovat CWT signálu $f(t)$ pomocí vlnky $\|\psi_{u,s}\| = 1$ jako

$$Wf(u,s) = \langle f, \psi_{u,s} \rangle = \int_{-\infty}^{\infty} f(t) \frac{1}{\sqrt{s}} \psi^* \frac{t-u}{s} \, \mathrm{d}t, \tag{2.8}$$

kde ψ^* znamená komplexně sdruženou vlnku, pro reálné se nepočítá. Také se zde nachází dilatace a translace vlnky, která se provádí pro každý výpočet integrálu, tzn. v podstatě se počítá konvoluce pro každou deformovanou vlnku. Protože máme normalizaci definovanou již v rovnici vlnky (2.7), můžeme rovnici CWT zjednodušit na

$$Wf(u,s) = \int_{-\infty}^{\infty} f(t) \psi^*_{u,s} \, \mathrm{d}t. \tag{2.9}$$

Můžeme tedy říci, že se porovnává signál $f(t)$ s nekonečně mnoha dilatovanými vlnkami po nepřerušovaných časových úsecích a výsledkem je scalogram, který zobrazuje závislosti měřítka vlnky na čase (korelace signálu s vlnkou na různých měřítkách).

2.4 Diskrétní vlnková transformace

Protože se CWT pro svoje vlastnosti (nespojitost v měřítku, tak i v čase ⇒ vysoká redundance) nelze použít v praxi, musíme naše rovnice převést do diskrétního světa. Začneme u diskrétní vlnky:

$$\psi_{j,k}(t) = \frac{1}{\sqrt{s_0^j}} \psi \left(\frac{t - k u_0 s_0^j}{s_0^j} \right), \tag{2.10}$$

kde j a k jsou celá čísla, která se starají o dilataci a translaci (protože jsou to čísla celá, skáče se po krocích), $s_0 > 1$ je pevně daný dilatační krok a $u > 0$ určuje pozici. Z rovnice

také vyplývá, že velikost kroku časového posunu $u_0 s_0^j$ je přímo úměrná měřítku s_0^j. Teď si můžeme rovnici vlnky spojit s DWT pro spojitý signál $x(t)$:

$$Wf(j,k) = \int_{-\infty}^{\infty} x(t) \frac{1}{s_0^{j/2}} \psi\left(s_0^{-j} t - k u_0\right) \mathrm{d}t, \tag{2.11}$$

kde j a k nám indexují pozici v mřížce získaných koeficientů transformace $Wf(j,k)$, tzv. *detailových koeficientů*. Pokud zvolíme kroky $s_0 = 2$ a $u_0 = 1$ (po úpravě):

$$\psi_{j,k}(t) = 2^{-j/2} \psi(2^{-j} t - k), \tag{2.12}$$

dilatace a translace se budou měnit po mocninách dvou a vlnkové rámce se nebudou překrývat (frekvenčně i časově), tzn. transformační koeficienty se nebudou duplikovat (vytváří *ortonormální bázi*). V tomto případě se jedná o tzv. *dyadickou mřížku*, která nám umožňuje dokonale transformovat signál, tzn. bez nadbytečných (opakujících se) či ztracených informací. Potom se nám rovnice DWT [3] přepíše na

$$Wf(j,k) = \int_{-\infty}^{\infty} x(t) \psi_{j,k}(t) \, \mathrm{d}t. \tag{2.13}$$

Tímto způsobem transformace lze vstupní signál pomocí koeficientů DWT a *inverzní DWT* perfektně rekonstruovat zpět:

$$x(t) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} Wf(j,k) \psi_{j,k}(t). \tag{2.14}$$

2.4.1 Měřítková funkce

Dyadická měřítková funkce nám mění měřítko signálu či vlnky s ohledem na zachovaní energie (normalizování). Má tvar:

$$\phi_{j,k}(t) = 2^{-j/2} \phi(2^{-j} t - k), \tag{2.15}$$

tedy stejný jako vlnka (2.12) (někdy označována, jako *otcovská vlnka*), její vlastnost je ovšem jiná:

$$\int_{-\infty}^{\infty} \phi_{0,0}(t) \, \mathrm{d}t = 1, \tag{2.16}$$

od té uvedené v (2.6). Tato měřítková funkce je ortogonální pouze sama na sebe v translaci, ale není již v dilataci, tzn. že konvolucí se signálem $x(t)$ získáme *aproximační koeficienty* a to obdobně jako u DWT (2.13):

$$a(j,k) = \int_{-\infty}^{\infty} x(t) \phi_{j,k}(t) \, \mathrm{d}t. \tag{2.17}$$

Funkce svým způsobem vypočítává vážené průměry ze vstupního signálu o daném úseku (měřítku) j. Vypočítaným koeficientům se také říká *diskrétní aproximace* signálu pouze jednoho daného měřítka j. Pro výpočet *spojité aproximace* signálu daného měřítka stačí sečíst měřítkové funkce a dané koeficienty diskrétní aproximace:

$$x_j(t) = \sum_{-\infty}^{\infty} a(j,k) \phi_{j,k}(t). \tag{2.18}$$

Díky těmto skutečnostem můžeme vyjádřit signál $x(t)$ součtem aproximačních a detailových (vlnkových) koeficientů jako

$$x(t) = \sum_{k=-\infty}^{\infty} a(j_0, k)\phi_{j_0,k}(t) + \sum_{j=-\infty}^{j_0} \sum_{k=-\infty}^{\infty} Wf(j,k)\psi_{j,k}(t). \tag{2.19}$$

Vídíme, že je signál $x(t)$ vyjádřen kombinací součtu aproximace sebe samotného o daném měřítku j_0 a součtu posloupností detailů signálu od měřítka $j = -\infty$ po j_0. Detail signálu o daném měřítku j je definován jako

$$d_j(t) = \sum_{k=-\infty}^{\infty} Wf(j,k)\psi_{j,k}(t), \tag{2.20}$$

pak můžeme rovnici (2.19) upravit do tvaru

$$x(t) = x_{j_0}(t) + \sum_{k=-\infty}^{j_0} Wf(j,k)\psi_{j,k}(t). \tag{2.21}$$

Tato rovnice nám popisuje signál (na jedné úrovni měřítka j), který může být vyjádřen sám sebou na úrovni s nižším měřítkem $j-1$ (s vyšším rozlišením) spolu s detaily signálu na stejném měřítku. Této zavislosti se říká *multirozlišovací reprezentace*:

$$x_{j-1}(t) = x_j(t) + d_j(t). \tag{2.22}$$

2.4.2 Měřítková rovnice (dilatační rovnice)

Dilatační rovnice popisuje funkci měřítka $\phi(t)$, jako zúženou a posunutou verzi sebe samé

$$\phi(t) = \sum_n c_n \phi(2t - n), \tag{2.23}$$

kde $\phi(2t-k)$ je zúžená verze funkce $\phi(t)$, která je posunutá na časové ose o k (celé číslo) a která je ovlivněná měřítkových koeficientem c_k. Tento koeficient musí splňovat podmínku:

$$\sum_n c_n = 2. \tag{2.24}$$

Pro vytvoření ortogononálního systému musí platit:

$$\sum_n c_n c_{n+2n'} = \begin{cases} 2 & n' = 0 \\ 0 & n' \neq 0 \end{cases}. \tag{2.25}$$

S využitím těchto vlastností můžeme vyjádřit rovnici vlnky $\psi(t)$ takto:

$$\psi(t) = \sum_n (-1)^n c_{N_n - 1 - n} \phi(2t - n), \tag{2.26}$$

která nám zaručuje ortogonalitu mezi vlnkami a jejími měřítkovými funkcemi pro konečný počet měřítkových koeficientů N_k (pro interval $[0, N_k - 1]$). Nakonec pomocí rovnic (2.15)

a (2.23) zapíšeme výslednou měřítkovou funkci vyjádřenou sebe samou z následujícího menšího měřítka s ohledem na jejich koeficienty:

$$\phi_{j+1,k}(t) = \frac{1}{\sqrt{2}} \sum_n c_n \phi_{j,2k+n}(t), \tag{2.27}$$

a to samé lze zapsat pro vlnkovou funkci:

$$\psi_{j+1,k}(t) = \frac{1}{\sqrt{2}} \sum_n b_n \phi_{j,2k+n}(t), \tag{2.28}$$

kde

$$b_n = (-1)^n c_{N_n-1-n} \tag{2.29}$$

je vlnkový koeficient vyjádřený měřítkovými koeficienty.

2.4.3 Rychlá vlnková transformace

Tato transformace je založena na principu uvedeném v minulé kapitole, tzn. na principu výpočtu aproximačních a vlnkových (detailových) koeficientů z předešlého výpočtu (aproximačních koeficientů) o větším měřítku. Tomuto postupu se také říká *dekompoziční algoritmus*.

Nejprve do rovnice (2.17) dosadíme (2.27):

$$a(j+1,k) = \int_{-\infty}^{\infty} x(t) \left[\frac{1}{\sqrt{2}} \sum_n c_n \phi_{j,2k+n}(t) \right] \mathrm{d}t, \tag{2.30}$$

tu můžeme upravit:

$$a(j+1,k) = \frac{1}{\sqrt{2}} \sum_n c_n \left[\int_{-\infty}^{\infty} x(t) \phi_{j,2k+n}(t) \right] \mathrm{d}t. \tag{2.31}$$

Vidíme, že integrál v závorkách je rovnice pro aproximační koeficienty $a(j, 2k+n)$:

$$a(j+1,k) = \frac{1}{\sqrt{2}} \sum_n c_n a(j,2k+n) = \frac{1}{\sqrt{2}} \sum_n c_{n-2j} a(j,k). \tag{2.32}$$

Teď už dokážeme vyjádřit aproximační koeficinety jednoho měřítka pomocí koeficientů z předešlého měřítka. To samé platí pro detailové koeficienty:

$$Wf(j+1,k) = \frac{1}{\sqrt{2}} \sum_n b_n a(j,2k+n) = \frac{1}{\sqrt{2}} \sum_n b_{n-2j} a(j,k). \tag{2.33}$$

Nyní známe postup jak vypočítat FWT, ale ještě nevíme jak rekonstruovat zpět transformovaný signál. Toho docílíme tak, že za rovnici (2.22), která popisuje signál o daném měřítku pomocí nižšího měřítka, dosadíme substituce odpovídající rovnice:

$$x_{j-1}(t) = \sum_n a_{j,k} \phi_{j,k}(t) + \sum_n Wf(j,k) \psi_{j,k}(t). \tag{2.34}$$

Když vyjádříme rovnice sebe samými pomocí předešlého měřítka, dostaneme:

$$x_{j-1}(t) = \sum_n a_{j,k} \frac{1}{\sqrt{2}} \sum_n c_{n-2k} \phi_{j-1,n}(t) + \sum_n Wf(j,k) \frac{1}{\sqrt{2}} \sum_n b_{n-2k} \phi_{j-1,n}(t). \tag{2.35}$$

Z rovnice (2.18) víme, že $x_j(t)$ můžeme vyjádřit pomocí aproximačních koeficientů stejného měřítka j. Poté můžeme upravit rovnici do konečného tvaru *rekonstrukčního algoritmu* jako

$$a(j-1,k) = \frac{1}{\sqrt{2}} \sum_n c_{k-2n} a(j,n) + \frac{1}{\sqrt{2}} \sum_n b_{k-2n} Wf(j,n). \tag{2.36}$$

2.4.4 Rychlá vlnková transformace pomocí filtrování

V předešlé sekci jsme si popsali tzv. *dekompoziční a rekonstrukční algoritmus*. Tyto algoritmy lze chápat jako pásmové propusti, tedy dolní propust a horní propust. Pak mluvíme o tzv. *dekompozičním a rekonstrukčním filtrování*, které lze vyjádřit konvolucemi [7]:

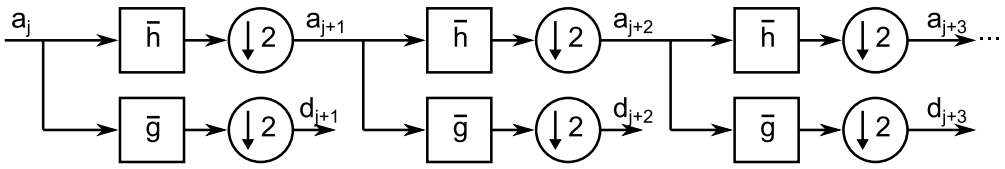
$$a_{j+1}[k] = \sum_{n=-\infty}^{\infty} h[n-2k]a_j[n] = a_j * \bar{h}[2k], \tag{2.37}$$

$$d_{j+1}[k] = \sum_{n=-\infty}^{\infty} g[n-2k]a_j[n] = a_j * \bar{g}[2k], \tag{2.38}$$

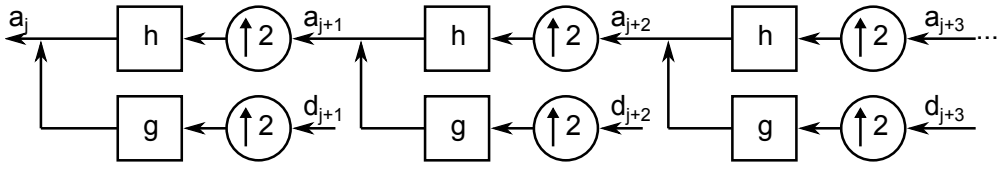
kde a_{j+1} a d_{j+1} jsou aproximační koeficienty daného stupně resp. detailové koeficienty daného stupně dekompozice. Pro rekunstrukci potom platí:

$$a_j[k] = \sum_{n=-\infty}^{\infty} h[n-2k]a_j[n] + \sum_{n=-\infty}^{\infty} g[n-2k]a_j[n]. \tag{2.39}$$

Při dekompozici se tedy vstupní signál, pomocí dolní a horní propusti, rozloží (odfiltrují) na aproximační (nižší freknvence) a detailové (vyšší frekvence) koeficienty a dále, než se provede další stupeň dekompozice, se signál z obou pásmových propustí podvzorkuje a zredukují se tak duplicitní koeficienty (každý sousední vzorek). Dále se provede opět filtrování pouze aproximačních koeficientů přes dolní a horní propust nižší úrovně. Tento iterační postup se provádí tak dlouho, dokud máme na vstupu dostatek vzorků.



(a) Dopředná kaskáda.



(b) Zpětná kaskáda.

Obrázek 2.5: Banka filtrů (2 stupně rozkladu).

Při rekonstrukci se provádí opačný postup. Koeficienty se nadvzorkují (mezi vzorky se vloží nuly) a pomocí obou pásmových propustí nejnižší úrovně se signál složí. V této chvíli jsou koeficienty obráceny a posunuty na konec signálu. Dále se provede opět nadvzorkování a filtrování tolikrát, dokud se nedojde až k nejvyšší úrovni. Poté je signál opět v původním tvaru.

2.4.5 Rychlá vlnková transformace dvojdimenzionálního signálu

V našem případě potřebujeme provést FDWT nad obrázkem, tzn. nad dvěma dimenzemi (výška, šířka). V tomto případě se tedy provádí konvoluce a podvzorkování čtyřikrát:

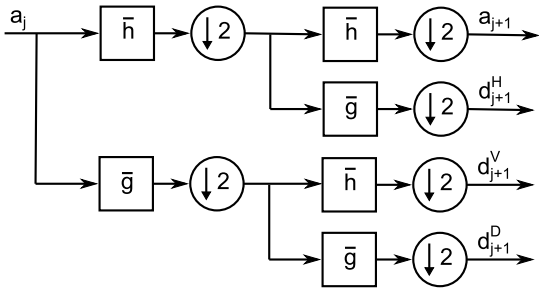
$$a_{j+1}[k] = a_j * \bar{h}\bar{h}[2k], \tag{2.40}$$

$$d^H_{j+1}[k] = a_j * \bar{h}\bar{g}[2k], \tag{2.41}$$

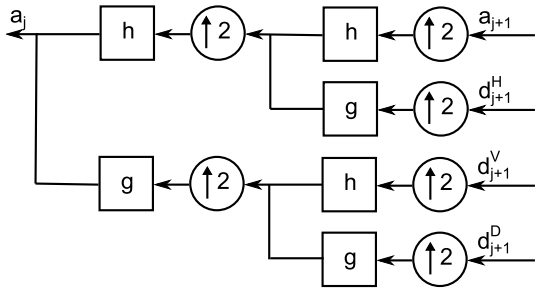
$$d^V_{j+1}[k] = a_j * \bar{g}\bar{h}[2k], \tag{2.42}$$

$$d^D_{j+1}[k] = a_j * \bar{g}\bar{g}[2k], \tag{2.43}$$

kde $k = (k_1, k_2)$ znamená souřadnice pixelu, dále pak v pořadí a_{j+1}, d^V_{j+1}, d^H_{j+1} a d^D_{j+1} jsou koeficienty aproximační, detailové pro řádky, sloupce a diagonálu.



(a) Dopředná kaskáda.



(b) Zpětná kaskáda.

Obrázek 2.6: Banka filtrů pro 2 rozměry (1 stupeň rozkladu).

2.4.6 Typy vlnek

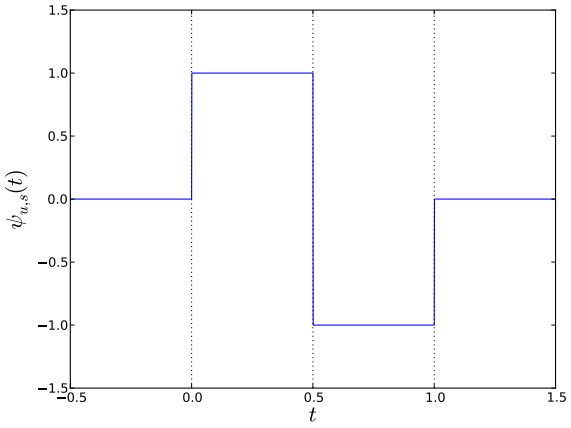
Existují stovky druhů vlnek a jejich názvy se odvozují od jejich specifických vlastností, a to především podle přípustnosti a tvaru (hladkost, symetrie, existence nosiče, atd.).

Vlnka Haar

Tato vlnka je jednou z nejstarších a také nejjednodušších. Skládá se pouze z několika bodů. Také se ji někdy říká vlnka Daubechies řádu 1. Je definována takto:

$$\psi(t) = \begin{cases} 0 & t < 0 \\ 1 & 0 \leq t < 1/2 \\ -1 & 1/2 \leq t < 1 \\ 0 & t > 1 \end{cases} \tag{2.44}$$

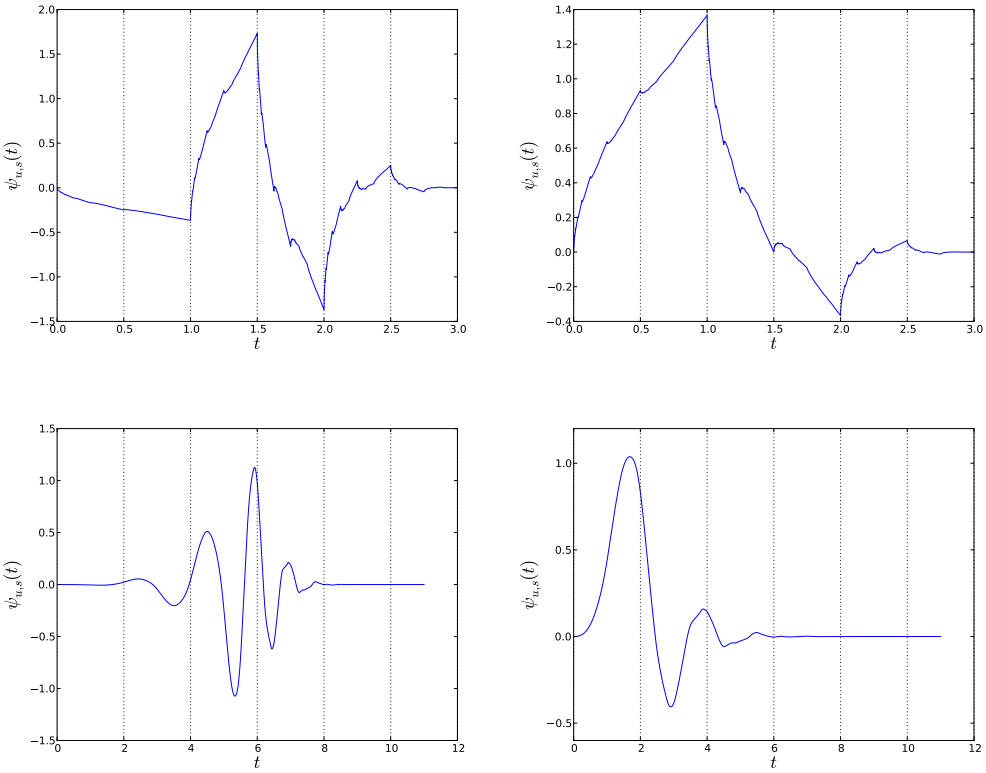
Má vlastnosti ortogonální i biorgonální a je i symetrická.



Obrázek 2.7: Vlnka Haar.

Vlnky Daubechies

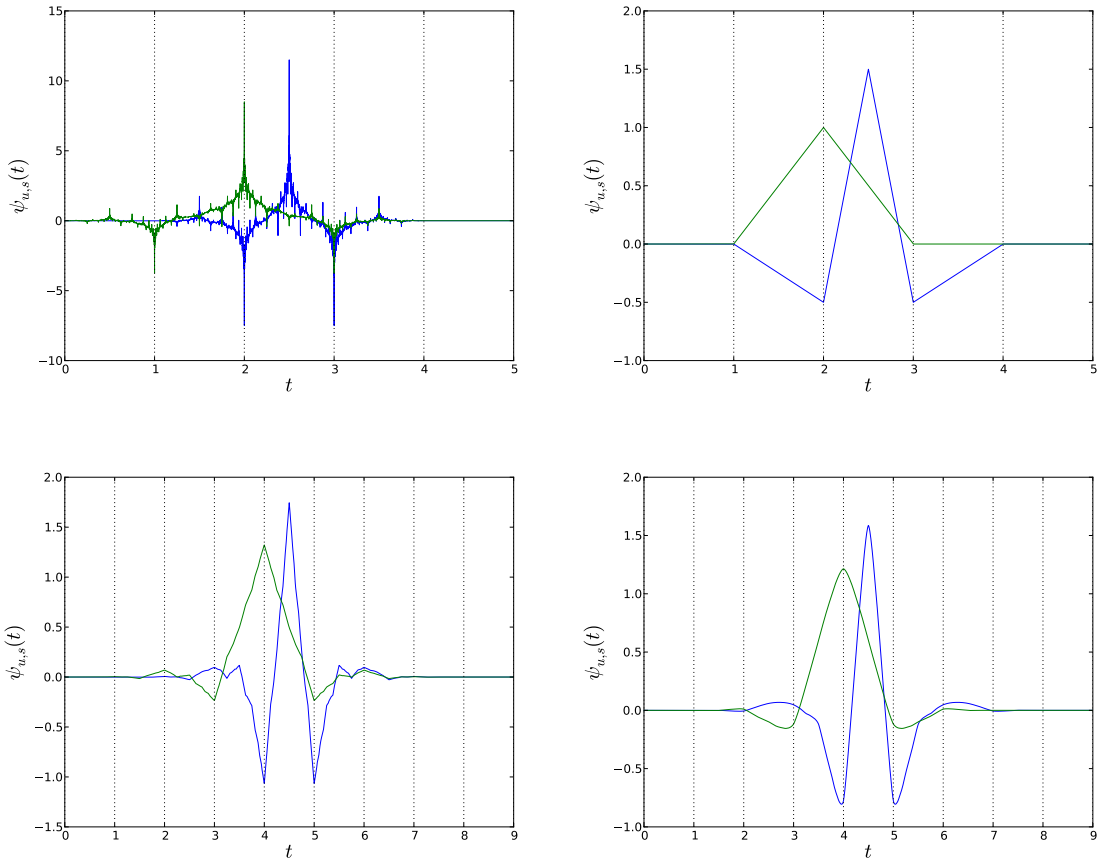
Vlnky Daubechies jsou rodinou ortogonálních, biortogonálních a asymetrických vlnek (kromě prvního stupně). Tato rodina má konečný počet měřítkových koeficientů, má tedy kompaktní nosič. Mají vhodné využití k potlačení, či získání polynomiální části signálu.



Obrázek 2.8: Daubechies vlnky db2, db6 a jejich měřítkové funkce.

Biortogonální spline vlnky

Biortogonální spline vlnky, nebo se jim také říká Cohen-Daubechies-Feauveau (CDF), jsou vlnky symetrické a biortogonální. Pro dekompozici a rekonstrukci používají odlišné vlnky, které mají různý počet nulových momentů. Pokud má tedy vlnka pro rekonstrukci méně nulových momentů než-li pri dekompozici, je výsledný signál hladší. Této vlastnosti se využívá např. při zpracování obrazu.



Obrázek 2.9: Biortogonální spline vlnka 2,2 (CDF 5/3), 4,5 (CDF 9/7) a jejich měřítkové funkce (spolu s reverzními vlnkami).

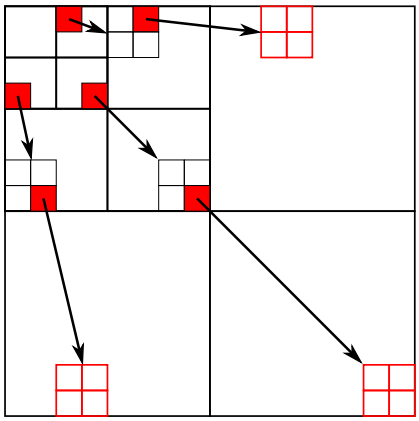
2.5 Kvantizace a komprese

Existují různé algoritmy pro kvantizaci a kompresi. Jeden z nejznámějších a nejstarších algoritmů je EZW (Embedded Zerotree Wavelet), který je dále modifikován a vylepšen, vznikl tak nový algoritmus SPIHT (Set Partitioning In Hierarchical Trees). Dalším algoritmem, který bude v této práci použit, je EBCOT (Embedded Block Coding with Optimal Truncation), který je využíván standardem JPEG 2000.

2.5.1 Embedded Zerotree Wavelet

Embedded Zerotree Wavelet [5] patří do rodiny tzv. *progresivního kódování*, tzn. jeho výstupem je posloupnost bitů, která lze kdekoli zastavit a sestavit tak z nich původní obraz se stejnou či nižší kvalitou. Lze tímto způsobem komprimovat obraz pro určitý kompresní poměr nebo velikost výsledných dat. Protože výstup tohoto algoritmu není dostatečně informačně zhuštěný, dále se kóduje pomocí entropického kódování (např. Huffmanovo nebo aritmetické).

Tento kodér využívá skutečnosti, že koeficienty dekompozičního obrazce lze chápat jako stromovou strukturu, kde každý následující uzel (*podpásmo*) bude obsahovat s velkou pravděpodobností menší hodnoty (postupným průchodem stromu lze provádět progresivní ukládání). Každé následující podpásmo je čtyřnásobně větší (v každém rozměru dvojnásobně), tak jak je popsáno na obrázku 2.10.



Obrázek 2.10: Dekompoziční obraz jako strom podpásem - prostorově orientační stromy.

Algoritmus prochází strom podpásem a rozděluje koeficienty na významné (Positive significant - POS, Negative significant - NEG) a nevýznamné (Isolated Zero - IZ, Zerotree Root - ZTR). Významnost či nevýznamnost se posuzuje podle prahu, který se po každém průchodu stromem snižuje na polovinu (proto práh musí být číslo o násobku dvou, které se určuje podle hodnoty nejvyššího koeficientu; práh má poté poloviční hodnotu). Je-li absolutní hodnota koeficientu větší jak daný práh, je označen jako POS (pro kladná) nebo NEG (pro záporná čísla). Pokud je koeficient nižší, pak je označen jako IZ (pokud alespoň jeden jeho následník obsahuje významný koeficient) nebo ZTR (pokud jsou všichni jeho následníci nevýznamní) a jeho hodnota je nulová. Takto zpracovaný koeficient se nazývá *rekonstrukční koeficient*. Pro další průchod se práh sníží o jeho poloviční hodnotu a koeficienty jsou přepočítány tak, že se v každém dalším kroku přibližuje původní hodnotě. Tento postup se opakuje, dokud není práh implicitně roven jedné. Pseudokód by mohl vypadat následovně [6]:

1. *Inicializace* - výběr prahu $T = T_0$, kde $|w(m)| < T_0$.
2. *Změna prahu* $T_k = T_{k-1}/2$
3. *Porovnávací průchod* - pro každou hodnotu $w(m)$ proveď:
   if $|w(m)| \geq T_k$ **then**

Zapiš znaménkový bit hodnoty $w(m)$
$w_Q(m) = T_k$
else if $|w(m)| < T_k$ **then**
$w_Q(m) = 0$
end if

4. *Upřesňovací průchod* - pro každou významnou hodnotu proveď prahování s prahem T_j, kde $j < k$ (pouze pokud $k > 1$):
if $|w(m)| \in |w_Q, w_Q(m) + T_k)$ **then**
Zapiš bit 0
else if $|w(m)| \in |w_Q(m) + T_k, w_Q(m) + 2T_k)$ **then**
Zapiš bit 1
$w_Q(m) = w_Q(m) + T_k$
end if

5. Vrať se zpět na krok 2.

Aritmetické kódování

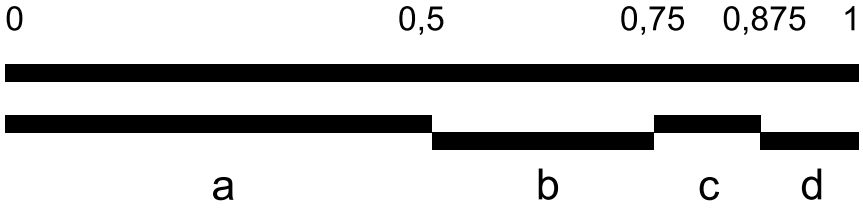
Toto entropické kódování je založeno na pravděpodobnostech výskytu určitých vzorů (např. znaků nebo posloupnost bitů), které se mají vyskytovat v dané množině, kterou chceme zakódovat. Pravděpodobnosti výskytu těchto vzorů jsou dány modelem a jejich součet musí být roven 1. Používané modely:

- *statický* - předem dané pravděpodobnosti, které se v průběhu kódovaní nemění,
- *order-n* - pravděpodobnosti se přepočítávají v průběhu kódovaní s ohledem na kontext předešlých výskytů v rozmezí daném n,
- *adaptivní* - pravděpodobnosti výskytu nejsou předem známy a vypočítávají se průběžně při kódování

Průběh kódování by mohl být následující. Mějme statický model M a abecedu $A = a, b, c, d$. Jejich pravděpodobnosti jsou definovány takto:

- $P_M(a) = 0{,}5$,
- $P_M(b) = 0{,}25$,
- $P_M(c) = 0{,}125$,
- $P_M(d) = 0{,}125$.

Potom je interval $\langle 0; 1)$ rozdělen na podintervaly:

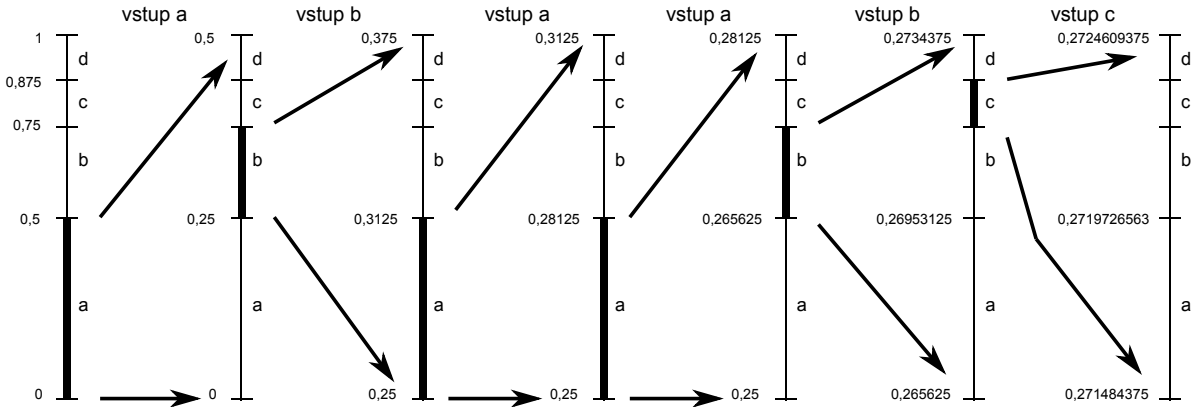


Obrázek 2.11: Rozdělení intervalu $\langle 0; 1)$ na podintervaly.

Pokud bychom kódovali zprávu $Z = abaabc$ pomocí výše uvedených pravděpodobností, postup by byl následující:

1. první znak je a, má pravděpodobnost $P_M(a) = 0{,}5$, počáteční interval $\langle 0; 1)$ je rozdělen na $\langle 0; 0{,}5)$ dle obrázku 2.11,
2. následující znak je b s pravděpodobností $P_M(b) = 0{,}25$, interval $\langle 0; 0{,}5)$ je rozdělen na $\langle 0{,}25; 0{,}375)$,
3. následující znak je a s pravděpodobností $P_M(a) = 0{,}5$, interval $\langle 0{,}25; 0{,}375)$ je rozdělen na $\langle 0{,}25; 0{,}3125)$,
4. následující znak je a s pravděpodobností $P_M(a) = 0{,}5$, interval $\langle 0{,}25; 0{,}3125)$ je rozdělen na $\langle 0{,}25; 0{,}28125)$,
5. následující znak je b s pravděpodobností $P_M(b) = 0{,}25$, interval $\langle 0{,}25; 0{,}28125)$ je rozdělen na $\langle 0{,}28125; 0{,}2734375)$,
6. následující znak je c s pravděpodobností $P_M(c) = 0{,}125$, interval $\langle 0{,}28125; 0{,}2734375)$ je rozdělen na $\langle 0{,}271484375; 0{,}2724609375)$.

Vidíme, že není využit znak d, pro který jsme si vyhradili pravděpodobnost, a není tak využit potenciál kódování na maximum. Toto je jedna z nevýhod statického modelu. Princip postupu kódování je znázorněn na obrázku 2.12.



Obrázek 2.12: Postup kódování pro $Z = abaabc$.

2.5.2 Set Partitioning In Hierarchical Trees

Tento algoritmus vylepšuje původní návrh EZW a modifikuje ho tak, že je jeho výstup dostatečně informačně zhuštěný a není tak potřeba dodatečného kódování. Zavádí 4 druhy množin [2]:

- $\mathcal{O}(i, j)$: indexy přímých potomků (offspring)
- $\mathcal{D}(i, j)$: indexy všech potomků (descendant), včetně přímých
- $\mathcal{H}$: obsahuje indexy nejvyšší úrovně rozkladu (highest pyramid level)
- $\mathcal{L}(i, j) = \mathcal{D}(i, j) - \mathcal{O}(i, j)$: indexy všech nepřímých potomků.

Množina $\mathcal{O}$ je pro dvě dimenze definována jako

$$\mathcal{O}(i,j) = (2i, 2j), (2i+1, 2j), (2i, 2j+1), (2i+1, 2j+1). \tag{2.45}$$

Stejně, tak jako je tomu u EZW, se koeficienty testují na významnost $S_n(\mathcal{T})$, kde $\mathcal{T}$ je množina indexů:

$$S_n(\mathcal{T}) = \begin{cases} 1 & \max\limits_{(i,j)\in\mathcal{T}} |c_{i,j}| \geq 2^n \\ 0 & jinak. \end{cases} \tag{2.46}$$

Dále SPIHT používá dělení množin a definuje jejich pravidla následovně:

1. Počáteční množina je tvořena množinami $\{(i,j)\}$ a $\mathcal{D}(i,j)$ pro všechna $(i,j) \in \mathcal{H}$.
2. Pokud je množina $\mathcal{D}(i,j)$ významná, pak je rozdělena na $\mathcal{L}(i,j)$ a čtyři samostatné prvky $(k,l) \in \mathcal{O}(i,j)$.
3. Pokud je množina $\mathcal{L}(i,j)$ významná, pak je rozdělena na čtyři množiny $\mathcal{D}(k,l)$ s $(k,l) \in \mathcal{O}(i,j)$.

Pseudokód algoritmu SPIHT:

1. *Inicializace*
   $n = \lfloor log_2(max_{(i,j)\{|i,j|\}}) \rfloor$
   LSP = prázdná
   $LIP = \mathcal{H}$
   LIS = přímí potomci z $\mathcal{H}$
2. *Porovnávací průchod* - pro každou hodnotu $(i,j) \in LIP$ proveď:
   Zapiš $S_n(i,j)$
   if $S_n(i,j) == 1$ **then**
   Přesuň (i,j) do LSP
   Zapiš znaménko $c_{i,j}$
   end if

   2.1. Pro každou hodnotu $(i,j) \in LIS$ proveď:

   2.1.1. *Zpracování prvku typu A*

   if (i,j) je typu A **then**
   Zapiš $S_n(\mathcal{D}(i,j))$
   if $S_n(\mathcal{D}(i,j)) == 1$ **then**
   for all $(k,l) \in \mathcal{O}(i,j)$ **do**
   Zapiš $S_n(k,l)$
   if $S_n(k,l) == 1$ **then**
   Přidej (k,l) do LSP
   Zapiš znaménko $c_{i,j}$
   else if $S_n(k,l) == 0$ **then**
   Přidej (k,l) na konec LIP
   end if
   end for
   if $\mathcal{L}(i,j) \neq$ prázdná **then**

Přesuň (i, j) na konec LIS jako by to byl prvek typu B
běž na krok **2.1.2**
else
Odeber z LIS prvek (i, j)
end if
end if
end if

2.1.2. *Zpracování prvku typu B*

if (i, j) je typu B **then**
Zapiš $S_n(\mathcal{L}(i, j))$
if $S_n(\mathcal{L}(i, j)) == 1$ **then**
Přidej každý $(k, l) \in \mathcal{O}(i, j)$ na konec LIS jako by to byl prvek typu A
Odeber (i, j) z LIS
end if
end if

3. *Upřesňovací průchod* - pro každý prvek $(i, j) \in LSP$, kromě těch prvků přidaných v posledním srovnávacím kroku, zapiš n-tý nejvýznamější bit $|c_{i,j}|$.

4. Dekrementuj n a vrať se zpět na krok 2.

2.5.3 Embedded Block Coding with Optimal Truncation Points

Embedded Block Coding with Optimal Truncation Points, neboli zkráceně EBCOT [4][1], byl vyvíjen pro standard JPEG2000. Tento algoritmus je přímo komplexním nástrojem pro zpracování obrazu. Obsahuje takové vymoženosti jako jsou např. rotace obrazu bez opětovné komprese nebo dekomprese pouze vybrané části (výřezu) obrazu.

Jak název napovídá, obraz je rozdělen do bloků (typicky 32×32 až 64×64 pixelů), které jsou samostatně zpracovány ve 2 úrovních:

1. *první úroveň (Tier-1)* dělí koeficienty transformace na podbloky (16×16), které jsou vyjádřeny tzv. *quad-tree* strukturou (každý uzel má 4 následníky; podobně jako na obrázku 2.10) a následně se kóduje pro každou bitovou rovinu 2^p pomocí čtyř průchodů:

   1.1. *Forward Significance Propagation Pass* - prochází koeficienty a kóduje je pomocí *ZC* (*Zero Coding*) nebo *RLC* (*Run-Length Coding*) primitiv. Pokud je koeficient významný, tj. $\upsilon_i^p(k_1, k_2) \geq 2^p$ ($\upsilon_i^p(k_1, k_2)$ je kvantizovaný koeficient), provede se navíc *SC* (*Sign Coding*), který předpovídá znaménko podle kontextu svých sousedících koeficientů.

   1.2. *Reverse Significance Propagation Pass* - to samé jako 1. průchod s tím, že se prochází v opačném pořadí v případě, že se změnila významnost některých koeficientů.

   1.3. *Magnitude Refinement Pass* - kóduje již významné koeficienty pomocí *MR* (*Magnitude Refinement*) primitiv.

   1.4. *Normalization Pass* - tento krok se provede pro ty koeficienty (týká se i těch nevýznamných), které nebyly zakódovány v předchozích třech krocích. Zde se použijí RLC a SC primitiva.

2. *Druhá úroveň (Tier-2)* pracuje s informacemi bloků poskytnuté první úrovní a vytváří z nich tzv. *quality layers (vrstvy kvality)*. Tyto vrstvy nám říkají, jak moc je v daném bloku nadbytečných informací. Každá vrstva je číslována od 1, kde vyšší číslo znamená vyšší úroveň kvality.

Kódovací primitiva jsou definována takto:

- *Zero Coding (ZC)* - je použit pro ty koeficienty, kde není možné použít RLC. Kóduje se pomocí kontextu sousedů:
  - *horizontální* $h_i(k_1, k_2) = \sigma_i(k_1 + 1, k_2) + \sigma_i(k_1 - 1, k_2)$, kde $0 \leq h_i(k_1, k_2) \leq 2$,
  - *vertikální* $v_i(k_1, k_2) = \sigma_i(k_1, k_2 + 1) + \sigma_i(k_1, k_2 - 1)$, kde $0 \leq v_i(k_1, k_2) \leq 2$,
  - *diagonální* $h_i(k_1, k_2) = \sigma_i(k_1+1, k_2)+\sigma_i(k_1-1, k_2)+\sigma_i(k_1, k_2+1)+\sigma_i(k_1, k_2-1)$, kde $0 \leq d_i(k_1, k_2) \leq 4$.

  Pro vyhodnocení se použije tabulka 2.1.

LL, LH a HL pásmo				HH pásmo		
$h_i(k_1, k_2)$	$v_i(k_1, k_2)$	$d_i(k_1, k_2)$	Oznaceni	$d_i(k_1, k_2)$	$h_i(k_1, k_2) + v_i(k_1, k_2)$	Oznaceni
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	0	> 1	2	0	> 1	2
0	1	x	3	1	0	3
0	2	x	4	1	1	4
1	0	0	5	1	> 1	5
1	0	> 0	6	2	0	6
1	> 0	x	7	2	> 0	7
2	x	x	8	> 2	x	8

Tabulka 2.1: Přiřazení kontextu koeficientu pomocí ZC.

- *Run-Length Coding (RLC)* - je použit, pokud jsou splněna následující pravidla:
  1. čtyři po sobě jdoucí koeficienty musí být nevýznamné
  2. a zároveň musí mít nevýznamné i sousedy,
  3. musí také být ve stejném podbloku,
  4. řádkový index k_1 prvního koeficientu musí být sudý.

  Pokud jsou splněna tato pravidla a jeden ze čtyř těchto koeficietů je označen jako významný, pro zakódování je použit jediný znak (pořadí daného koeficientu).

- *Sign Coding (SC)* - je použit pouze jednou na každý významný koeficient v dané bitové rovině. Znaménko se určuje pravidly:
  - $\bar{h}_i(k_1, k_2) = 0$, pokud oba horizontální sousedé jsou nevýznamní nebo jsou oba významní s rozdílnými znaménky,
  - $\bar{h}_i(k_1, k_2) = 1$, pokud alespoň jeden horizontální soused je kladný
  - $\bar{h}_i(k_1, k_2) = -1$, pokud alespoň jeden horizontální soused je záporný.

Pro $\bar{v}_i(k_1, k_2)$ platí ta samá výše uvedená pravidla (ve vertikálním směru). Poté se vypočítá znaménko vzorcem:

$$\chi_i(k_1, k_2) = AC(Oznaceni) \oplus XORbit, \tag{2.47}$$

kde $\chi_i(k_1, k_2)$ je znaménko, AC je aritmetický kodér, $Oznaceni$ a $XORbit$ jsou proměnné z tabulky 2.2.

$\bar{h}_i(k_1, k_2)$	$\bar{v}_i(k_1, k_2)$	Oznaceni	XORbit
1	1	4	0
1	0	3	0
1	-1	2	0
0	1	1	0
0	0	0	0
0	-1	1	1
-1	1	2	1
-1	0	3	1
-1	-1	4	1

Tabulka 2.2: Přiřazení znaménkového kontextu koeficientu pomocí SC.

- *Magnitude Refinement (MR)* - pravidla pro kódování jsou dána tabulkou 2.3. Proměnná $\bar{\sigma}_i(k_1, k_2)$ označuje stav (0 nebo 1) použití MR. V případě, že již byla dříve aplikována MR primitiva, obsahuje stav 1. V opačném případě obsahuje stav 0.

$\bar{h}_i(k_1, k_2) + \bar{v}_i(k_1, k_2)$	$\bar{\sigma}_i$	Oznaceni
0	0	0
≥ 1	0	1
x	1	2

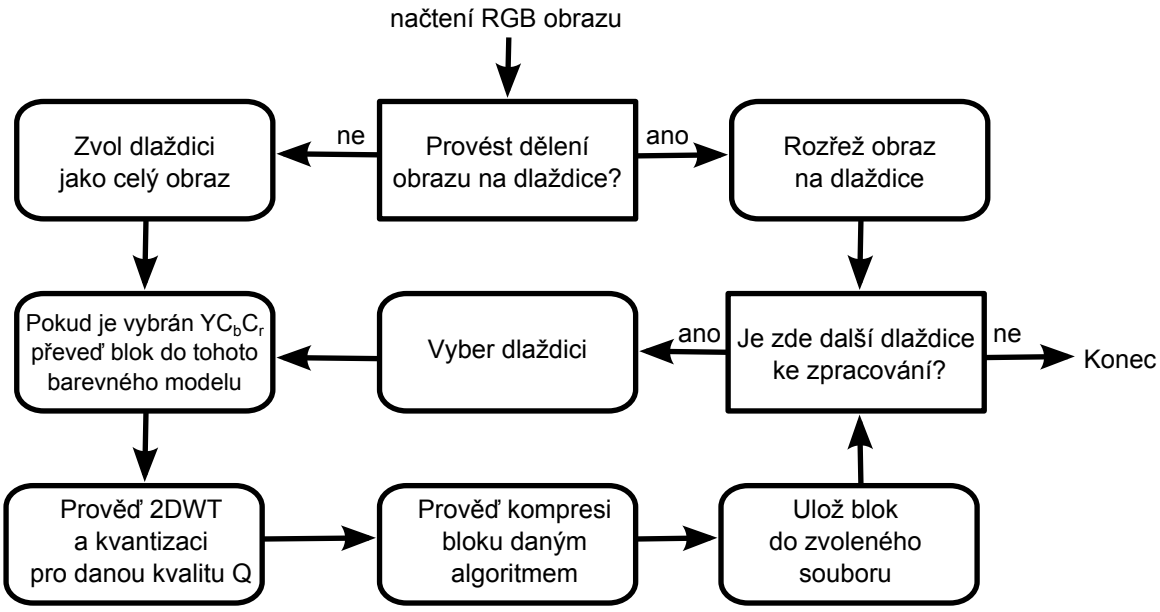
Tabulka 2.3: Přiřazení kontextu koeficientu pomocí MR.

Kapitola 3

Implementace

Tato část popisuje použité postupy a knihovny při implementaci ztrátové komprese obrazu. V první části této kapitoly se podíváme na předzpracování obrazu, tedy na to, co je potřeba provést před samotnou kopresí. Dále probereme implementaci diskrétní vlnkové transformace a na závěr samotné kompresní metody.

Zjednodušený postup komprese obrazu je na obrázku 3.1.



Obrázek 3.1: Zjednodušený diagram komprese obrazu.

Knihovna, vytvořena pro tuto práci, je napsána pro platformu linux s překladačem gcc verze 4.7.2.

3.1 Předzpracování

Před samotnou vlnkovou transformací je nejprve potřeba obraz určitým způsobem zpracovat. Chceme-li obraz zpracovávat po blocích (dlaždicích), musíme ho nejprve rozdělit na dané části a každou pak zpracovat samostatně. Tento krok lze přeskočit (vybere se jeden velký blok přes celý obraz) a následně přejít k další části předzpracování obrazu, k převodu RGB složek do $Y'C_bC_r$.

3.1.1 Rozřezání obrazu na dlaždice

Při dělení obrazu na dlaždice lze zvolit libovolnou velikost dané dlaždice. V tomto projektu je použit interval $\langle 8; 128 \rangle$ pixelů (lze tedy hodnota intervalu uložit do 8 bajtové proměnné), kde každá zvolená velikost je mocninou dvou. Takto zvolená hodnota je pak stranou čtverce jednoho bloku (např. 32×32 pixelů). Rozřezání obrazu je pak provedeno tak, jak je zobrazeno na obrázku 2.3.

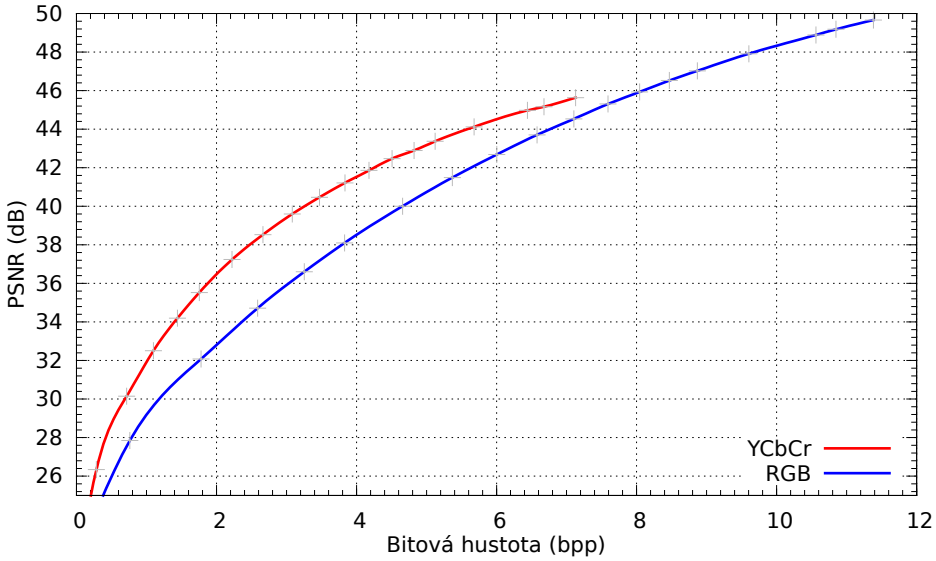
Problém při tomto postupu nastává tehdy, když nám bloky na okraji obrazu přesahují (viz. obr. 2.3). Tento problém je řešen až při samotné vlnkové transformaci obrazu, kdy se přesahující koeficienty zaplní nulami.

Výhodou tohoto postupu ovšem je, že lze každou dlaždici zpracovávat nezávisle na sobě, tzn. lze využít paralelního zpracování a zkrátit tak potřebnou dobu komprimace na systémech k tomu určeným (tento projekt se paralelizací nezabývá). Nevýhodou pak je zhoršení kvality komprese obrazu z důvodu nadbytečných informací, např. velký počet hlaviček pro každou dlaždici. Další důvod zhoršení kvality komprese se skrývá v nevyužitém potenciálu algoritmu EZW a SPIHT, který dosahuje dobré komprimace právě z důvodu velkých počtů nulových koeficientů v daném podstromu, které se při použití dlaždic velice zkracují a tím pádem se méně efektivněji komprimují. Tento rozdíl je znázorněn na obrázku 3.2.

Obrázek 3.2: Graf kvality komprese obrazu „beach1024“ (příloha A.1a) bez dlaždic a s dlaždicemi o velikosti 16×16 pixelů. Použitá vlnka je CDF 9/7 a algoritmus SPIHT.

3.1.2 Převod RGB do $Y'C_bC_r$

Abychom dosáhli lepšího komresního poměru, převedeme každý pixel dané dlaždice z barevného modelu RGB do více výhodného barevného modelu $Y'C_bC_r$ (rovnice 2.1). Z grafu 3.3 můžeme vyčíst, že jsou zde v některých případech značné rozdíly v kvalitě komprese pro daný barevný model, zejména pro přírodní fotografie, jakým je obraz „beach1024“ (příloha A.1a).



Obrázek 3.3: Vliv použitého barevného modelu na kvalitu komprese obrazu „beach1024“ (příloha A.1a). Použitá vlnka je CDF 9/7 a algoritmus SPIHT.

3.2 2D diskrétní vlnková transformace

Nyní máme obraz rozdělen na několik malých bloků, či jeden velký blok, a převeden do barevného modelu $Y'C_bC_r$. Dále je potřeba danou dlaždici transformovat pomocí 2DWT. K tomuto účelu je použita knihovna `blitzwave`. Její součástí je také možnost načtení obrazu typu `ppm` nebo `pnm` do paměti. Tato knihovna vyžaduje mít, ke své funkčnosti, nainstalovanou knihovnu `Blitz++`, která je potřebná pro zpracování vícedimenzionálních matic.

3.2.1 Knihovna blitzwave

`Blitzwave` používá pro načtení `pnm` obrazu funkci `void readPNM(const std::string &filename, blitz::Array<unsigned char, 3> &target)` a pro jeho uložení pak `void writePNM(const std::string &filename, blitz::Array<unsigned char, 3> &source)`. Takto načtený obraz je pak uložen v matici o třech dimenzích, kde postupně každá z nich obsahuje barevné kanály RGB.

Pro DWT zde máme třídu `bwave::WaveletDecomp` a pro samotnou transformaci pak její metodu `WaveletDecomp::apply(blitz::Array<tp_Type,tp_rank> &data)`. Dále, abychom provedli dekompozici obrazových koeficientů, je potřebujeme podvzorkovat následujícím kódem:

```
blitz::Array<blitz::TinyVector<int,3>, 1>
        idx(decomp.indices(imgArr));

for (int ix=0; ix<idx.rows(); ++ix)
  decomp.coeffs(imgSep, idx(ix)) = decomp.coeffs(imgArr, idx(ix));
```

Zdrojový kód 3.1: Dopředná dekompozice koeficientů obrazu.

```
for (int ix=0; ix<idx.rows(); ++ix)
  decomp.coeffs(imgArr, idx(ix)) = decomp.coeffs(imgSep, idx(ix));
```

Zdrojový kód 3.2: Zpětná dekompozice koeficientů obrazu.

kde `imgArr` (třída `blitz::Array`) obsahuje již transformovaná data a `imgSep` (třída `blitz::Array`) se vyplní dekompozičním obrazcem, `decomp` je třída `bwave::WaveletDecomp` a `idx` obsahuje indexy koeficientů.

Při zpětné dekompozici se pouze provede opačný postup. Zamění se matice `imgArr` za matici `imgSep` a naopak, poté se aplikuje inverzní metoda `WaveletDecomp::applyInv (blitz::Array<tp_Type,tp_rank> &data)`.

Výběr vlnek

Výběr vlnek se provádí parametrem konstruktoru třídy `bwave::WaveletDecomp`. Samotná vlnka je další třída `bwave::Wavelet`, kterou lze vytvořit pomocí konstruktoru, např. takto:

```
Wavelet myWavelet("CDF(2,2)", sqrt(2.0), sqrt(2.0)/2,
    Wavelet::LiftingStep(Wavelet::LiftingStep::DUAL,
    0, 2, -1, -1),
    Wavelet::LiftingStep(Wavelet::LiftingStep::PRIMAL,
    -1, 4, 1, 1)
);
```

Zdrojový kód 3.3: Příklad konstruktoru vlnky.

První parametr tohoto konstruktoru je název vlnky, dalšími dvěma parametry jsou její normalizační konstanty a nakonec se zadávají liftingové kroky, kterých může být více než je zde uvedeno.

Knihovna `blitzwave` má již několik vlnek předdefinovaných, včetně vlnky Haar, CDF 9/7 nebo CDF 5/3.

3.3 Kvantizace a komprese

Kvantizace je provedena vynásobením koeficientů transformace konstantou q, která nabývá hodnot v intervalu $\langle 0{,}01; 1\rangle$:

$$a_q(x, y) = a(x, y)q. \tag{3.1}$$

Reverzní kvantizace je vyjádřena takto:

$$a(x, y) = \frac{a_q(x, y)}{q}. \tag{3.2}$$

Kvantizační konstanta nám v tomto případě určuje míru kvality s jakou se daný obraz komprimuje, tzn. snižuje hodnotu koeficientu a tím i počet bitů, které jsou potřeba na zakódování. Např., máme-li koeficient s hodnotou 120, potřebujeme na jeho úplné zakódování 8 bitů, včetně znaménka +. Pokud bychom tento koeficient upravili kvantizační konstantou o hodnotě 0.5, snížila by se jeho hodnota na hodnotu 60, a v tuto chvíli by bylo potřeba na zakódování 7 bitů. Při zpětné kvantizaci se obnový jeho původní hodnota na 120.

U koeficientů, které se po kvantizaci sníží pod úroveň, kterou nejsme schopni zakódovat (typicky o hodnotě menší jak 1), dochází ke ztrátě informací. U vlnkových koeficientů tato

ztráta dat znamená, že se v obraze ztrácejí detaily (koeficienty o nízkých hodnotách) a aproximují se daným průběhem vlnky. Obraz se tak, se zvyšujícím se poměrem komprese, stává hladším.

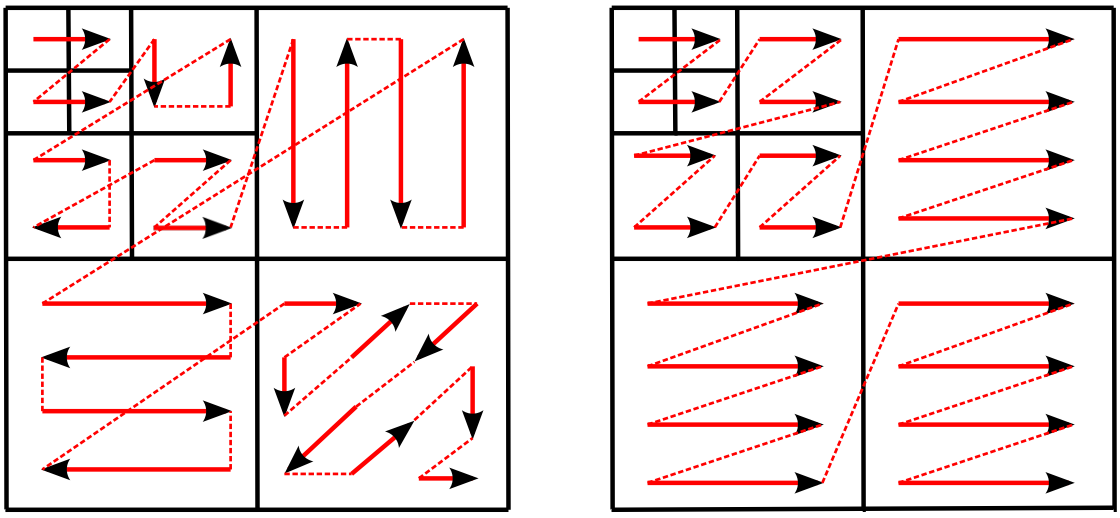
3.3.1 Embedded Zerotree Wavelet

Pro tento algoritmus je vyhrazena třída `EZW`. Jeho implementace se skládá z několika základních částí:

1. seznamy (důvody použití vektoru namísto seznamu jsou uvedeny v sekci 3.3.2):
   - `std::vector<int32_t> rafinList;` - obsahuje hodnoty koeficientů pro upřesňovací průchod,
   - `std::vector<ENUM_STATE> coderList;` - obsahuje zakódované stavy z upřesňovacího průchodu,
   - `std::vector<ENUM_STATE> sigpassList;` - obsahuje zakódované stavy z porovnávacího průchodu,
   - `std::string coderStr;` - obsahuje výstupní řetězec 0 a 1 z kodéru,
   - `std::vector<int32_t*> decoderList;` - obsahuje adresy významných koeficientů při dekódování.
2. Stavy `ENUM_STATE`:
   - EZW_NONE, EZW_POS, EZW_NEG, EZW_IZ, EZW_ZTR, EZW_0, EZW_1 s tím, že EZW_NONE je použit pro koeficienty, které nebyly dosud kódovány (tedy nejsou použity na výstupu z kodéru)
3. Pomocné metody:
   - `ENUM_STATE search_ztr_it(int16_t rowmin, int16_t colmin);` - hledá ZTR ve stromu pro daný koeficient, pokud ho najde vrátí ho, jinak IZ,
   - `void sigpass_step(int16_t myrow, int16_t mycol);` - provede jeden krok porovnávacího průchodu pro daný koeficient,
   - `void sigpass_optim();` - provede porovnávací průchod pro aktuální práh (průchod po rastrech),
   - `void rafinpass();` - provede upřesňovací průchod pro aktuální práh,
   - dále jsou zde ty samé metody pro dekódování s tím rozdílem, že obsahují slovo `decode`.
4. veřejné metody:
   - `std::string encode(std::vector< std::vector<int32_t> > &data, uint16_t & T0, float cratio);` - zakóduje matici a vrátí inicializační práh `T0`; `cratio` není implementováno; vrátí zakódovaný řetězec,
   - `std::vector< std::vector<int32_t> > decode (std::string &coder, int16_t x, int16_t y, uint16_t T0);` - dekóduje řetězec `coder` do matice o velikosti `x` a `y` a počátečním prahu `T0`.

Hlavní tělo kódovacího či dekódovacího algoritmu tvoří smyčka, která obsahuje metody `sigpass_optim()` a `rafinpass()` (v případě dekódování, jejich dekódovací verze), a po každém vykonání sníží práh `T` na polovinu.

Metoda `sigpass_optim()` se od metody `sigpass()` liší tím, že průchod podpásma neprovádí původně navrženým způsobem, ale tzv. průchodem po řádcích (obrázek. 3.4). Tento způsob průchodu zkracuje dobu komprimace přibližně o 30 až 40 procent a zjištěné rozdíly v kvalitě komprese jsou oproti původní verzi zanedbatelné (dokonce je nová verze lepší o několik bajtů při velikosti komprimovaného obrazu v řádů stovek až tisíců kB).



(a) Původní průchod.

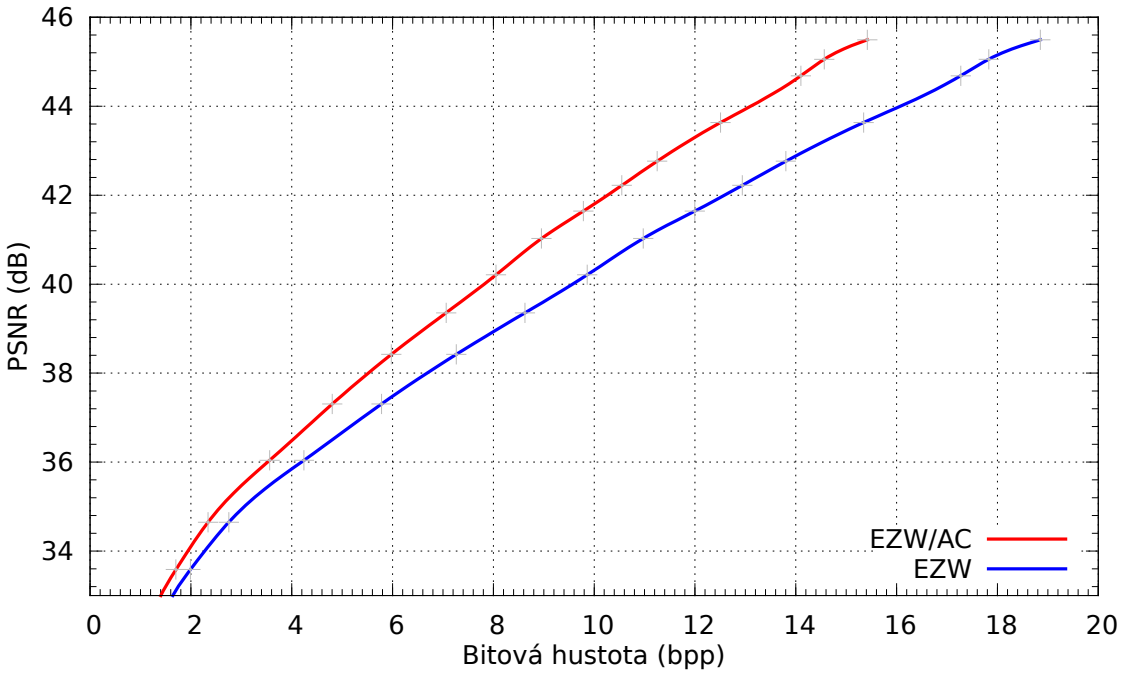
(b) Průchod po řádcích.

Obrázek 3.4: Různé způsoby průchodu podpásmy.

Aritmetický kodér

Po každém porovnávacím průchodu se zakódují uložené stavy v `coderList` statickým aritmetickým kodérem, který prvním průchodem seznamu vypočítá pravděpodobnosti těchto čtyř stavů a při druhém průchodu je zakódují a vrátí jako řetězec jedniček a nul.

Pro tento kodér je vyhrazena třída s názvem `ARITHMCODER`. Stavy zakóduje do bloku, který se skládá z hlavičky a zakódovaným dat. Hlavička obsahuje počet zakódovaných znaků, počet bajtů v bloku a pravděpodobnostní tabulku stavů. Příklad komprese s kodérem a bez kodéru je na obrázku 3.5.



Obrázek 3.5: Komprese obrazu „lena512“ (příloha A.1b)pomocí EZW s kodérem a bez kodéru. Použitá vlnka je CDF 9/7. Z grafu můžeme vyčíst, že aritmetický kodér zlepšil kompresní poměr přibližně o 15 až 20 procent.

3.3.2 Set Partitioning In Hierarchical Trees

Tento algoritmus je implementován podle původního návrhu [2], tzn. tak, jak je uvedeno v kapitole 2.5.2. Tam, kde bylo možné, byl použit objekt `std::vector`, protože se nad ním rychleji provádí některé operace, např. vkládání prvku na konec vektoru. Naopak při použití jiných operací je výhodnější použít objekt `std::list`, který má rychlejší vkládání a mazání prvků z různých pozic daného seznamu. Příklad těchto operací je znázorněn v tabulce 3.1.

Operace	`list` (čas [s])	`vector` (čas [s])
Vkládání prvku na konec	0,055	0,018
Vkládání prvku na začátek	0,069	35,000

Tabulka 3.1: Porovnání průměrných časů některých operací nad různými objekty s 500000 prvky.

1. Použité seznamy:
   - `std::list<int32_t> LIP;`,
   - `std::vector<int32_t> LSP;`,
   - `std::list<int32_t*> LIP_dec;` - používá se u dekódování. Má stejnou vlastnost jako LIP, s tím rozdílem, že se neukládá hodnota, ale adresa prvku v matici,
   - `std::vector<int32_t*> LSP_dec;` - podobné u předchozího řádku,

- `std::list<LIS_item> LIS;` - LIS_item obsahuje hodnotu a souřadnice koeficientu matice,
- `std::string coderVec;` - výstup z kodéru.

2. Pomocné metody:
   - `bool search_sig_it(int16_t rowmin, int16_t colmin);` - vrátí true, pokud najde významného potomka,
   - `void sortpass_step_A(int16_t myrow, int16_t mycol);` - provede jeden krok porovnávacího průchodu pro prvek typu A,
   - `void sortpass();` - porovnávací průchod,
   - `void rafinpass();` - upřesňovací průchod,
   - dále jsou zde metody pro dekódování, které provádí opačný postup k postupu zakódování.
3. Veřejné metody:
   - `std::string encode(std::vector< std::vector<int32_t> > &data, uint16_t &T0, float cratio);` - ta samá vlastnost jako v kapitole 3.3.1,
   - `std::vector< std::vector<int32_t> > decode(std::string &coder, int16_t x, int16_t y, uint16_t T0);` - ta samá vlastnost jako v kapitole 3.3.1.

Hlavní tělo, stejně tak, jak je tomu u EZW, kódovacího či dekódovacího algoritmu tvoří smyčka, která obsahuje metody `sortpass()` a `rafinpass()` (v případě dekódování jejich dekódovací verze), a po každém vykonání sníží práh `T` na polovinu.

3.3.3 Embedded Block Coding with Optimal Truncation Points

Tento algoritmus nebyl implementován celý. Byl použit první stupeň kódování Tier-1 s několika odlišnostmi od standardu [1]. V podstatě nebyly použity žádné jiné bloky, než codeblocks, které kódují vždy celé podpásmo. Minimální velikost bloku je 4×4 (tedy i podpásma). Každý blok se prochází po menších čtyř řádkových blocích tak, jak je tomu u standardu.

Každý blok se kóduje třemi průchody (*Significance propagation pass*, *Magnitude refinement pass*, *Cleanup pass*) s tím, že první průchod je vždy *Cleanup pass*. Tyto 3 průchody se provádí tak dlouho, dokud práh T nedosáhne nejnižší stanovené hodnoty.

Pro tento algoritmus zde máme vyhrazenu třídu `EBCOT` se 2 veřejnými metodami:

- `std::string encode(std::vector< std::vector<int32_t> > &data);`
- `std::vector< std::vector<int32_t> > decode(std::string &coder, int16_t x, int16_t y);`

MQ-Coder

Každý z těchto tří průchodů nám zakódují koeficienty kontextovými primitivy (sekce 2.5.3), které se dále kódují MQ-Coderem. MQ-Coder je modifikovaná verze adaptivního aritmetického kodéru Q-Coder. Tento kodér je podrobně popsán ve standardu [1].

Zde je pro něj vyhrazena třída `MQCODER`. Rozhraním tohoto objektu je tvořeno 3 veřejnými metodami:

- `void initDecoder(std::string &in);` - inicializace dekodéru,
- `std::string encode(std::vector<item_t> &coderList);` - zakódování jednoho bloku,
- `void decodeSymbol(item_t &s);` - dekódování jednoho symbolu,

kde item_t obsahuje 2 proměnné. První z nich je hodnota 0 nebo 1 a druhé je jedno z daných primitiv.

Výstupem metody `encode` je celý zakódovaný blok dat, který je následně obalen hlavičkou. O tento proces se stará sama třída `EBCOT` a to metodami:

- `void saveCb (std::string &outstr);` - uložení hlavičky s daty,
- `void loadCb (std::string &instr);` - načtení hlavičky a další obsluha.

Kapitola 4

Testy

V této části si porovnáme výsledky komprese při použití různých algoritmů (včetně JPEG a JPEG2000) a vlnek. Dále porovnáme vliv dlaždic na kvalitu obrazu a nakonec rozdíl kompresního poměru mezi barevným modelem RGB a $Y'C_bC_r$.

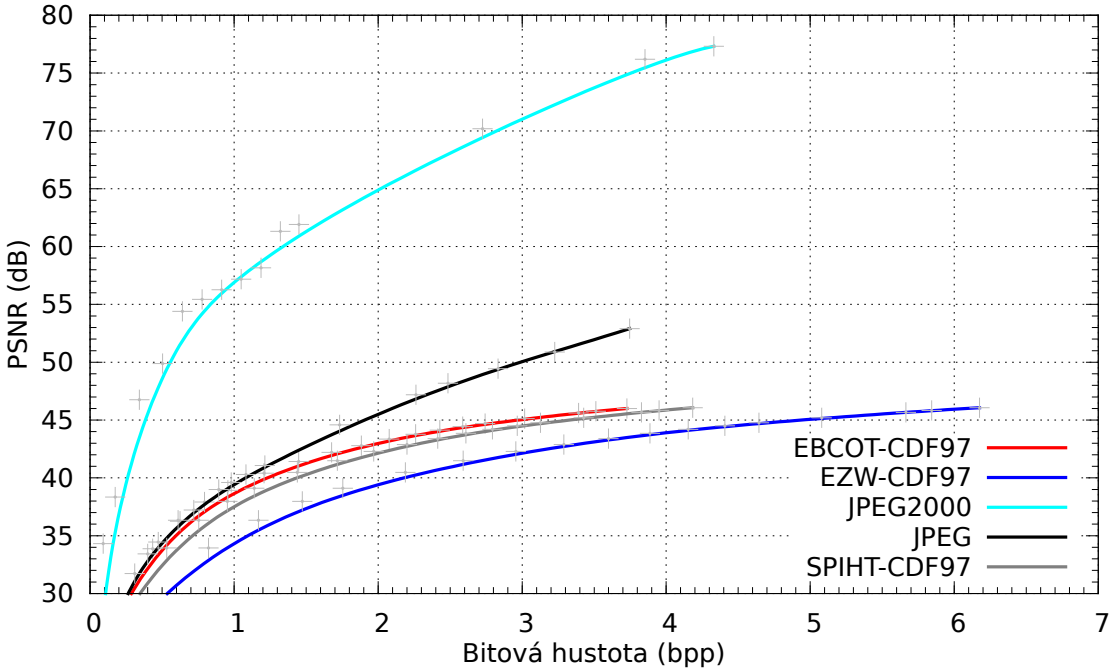
Zvolený postup je takový, že se každý druh testu provede nad sadou obrazů a výsledkem je aritmetický průměr křivek při určité kvalitě komprese. Použité obrázky jsou v příloze A.0.

Značení křivek je následující. Označení EZW je vždy algoritmus EZW s aritmetickým kodérem, a EBCOT vždy znamená algoritmus EBCOT s Tier1. Číslo uvedené za názvem algoritmu je vždy velikost dlaždice a text za tímto číslem značí použitou vlnku, např. EZW-128-HAAR znamená algoritmus EZW s aritmetických kodérem o velikosti bloku 128×128 pixelů a použité vlnce Haar. JPEG a JPEG2000 nemá žádné speciální značení.

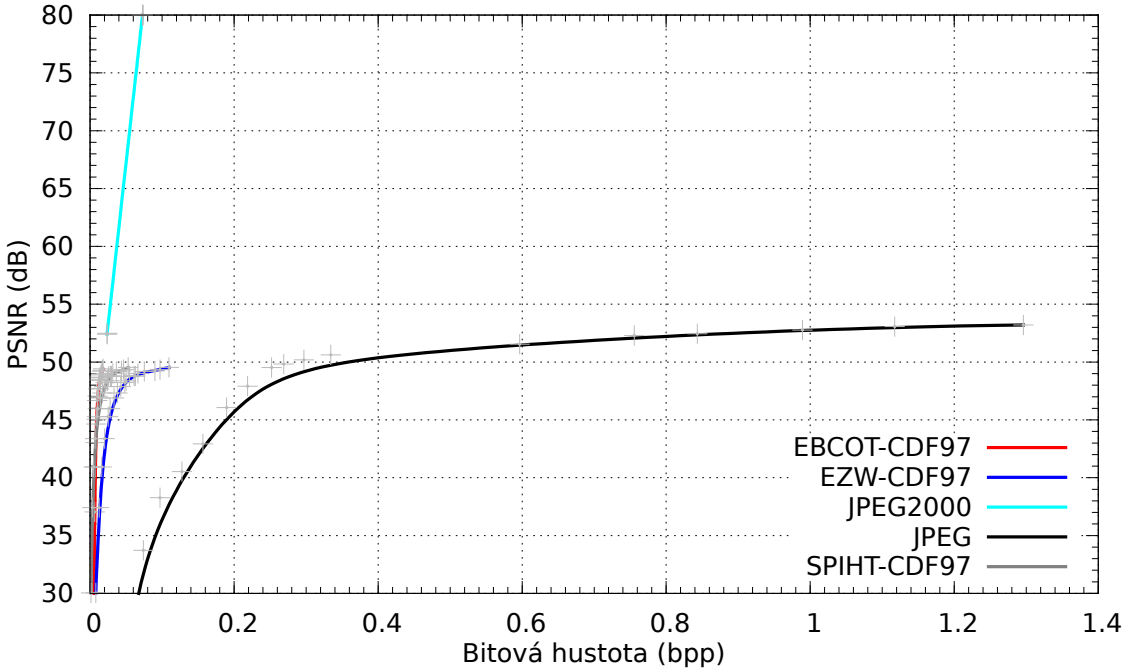
4.1 Algoritmy

Zde jsou zobrazeny grafy zmíněných algoritmů. Pro přehlednost byly testy provedeny nad jednou vlnkou CDF 9/7. Z grafu 4.1 vyplývá, že si průměrně nejlépe vedl algoritmus JPEG2000. Ve všech možných typech obrazů si vedl nejlépe.

Ovšem starší JPEG, který si vedl jako druhý nejlépe, si vždy tak dobře nevedl. Tam, kde se nachází hodně plynulých přechodů, jsou jeho výsledky nejhorší, např. na obrázku 4.2 vidíme, že si vlnkové algoritmy s hladkými přechody poradí velice dobře. Naopak, tam, kde nejsou téměř žádné přechody, se mu daří nejlépe (např. obrázek „cicle1024“ z přílohy A.0f) nebo tam, kde jsou velice ostré hrany, jako např. u obrazu „alphabet1024“ (příloha A.0e), B.1. Ale u přírodních obrazů, tam, kde jsou kombinace plynulých a rychlých přechodů, si vede většinou hůře než algoritmus SPIHT (obrázek „lena512“ z přílohy B.2). Další vybrané grafy jsou uvedeny v příloze B.



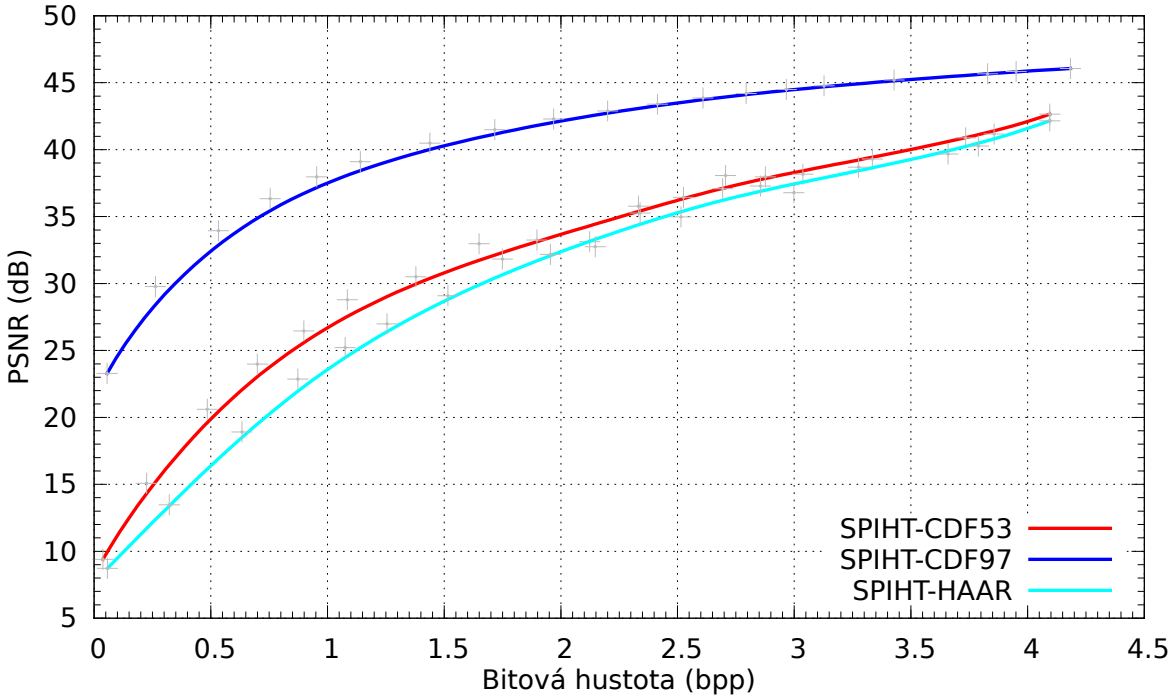
Obrázek 4.1: Graf vlivu použitého algoritmu na kvalitu komprese obrazu. Zobrazuje průměr křivek. Algoritmus JPEG dosahuje lepších výsledků právě z důvodu vyššího poměru použitých umělých obrázků.



Obrázek 4.2: Graf vlivu použitého algoritmu na kvalitu komprese obrazu „gradient1024“ (příloha A.0j). Zde vidíme, jakých excelentních výsledků dosahuje algoritmus JPEG2000. Při nejnižších kompresních poměrech dosahuje lepších výsledků než zbylé algoritmy při nejvyšší kvalitě.

4.2 Vlnky

Výběr vlnek hraje významnou roli v kompresi obrazu. U našeho souboru obrazů dosahuje průměrně nejlépe vlnka CDF 9/7. Ovšem v některých případech vykazuje lepších výsledků vlnka Haar než CDF 5/3, např.tomu tak je u obrazu „alphabet1024“ (příloha A.0e), výsledky jsou zobrazeny na grafu v příloze B.3. Je tomu tak, protože je zde velká jednobarevná plocha s mnoha pravoúhlými objekty, které dobře zapadají do nespojitostí této vlnky, obdobně dobře, ale ne zcela, si vede i u obrazu „cicle1024“ (příloha A.0e).



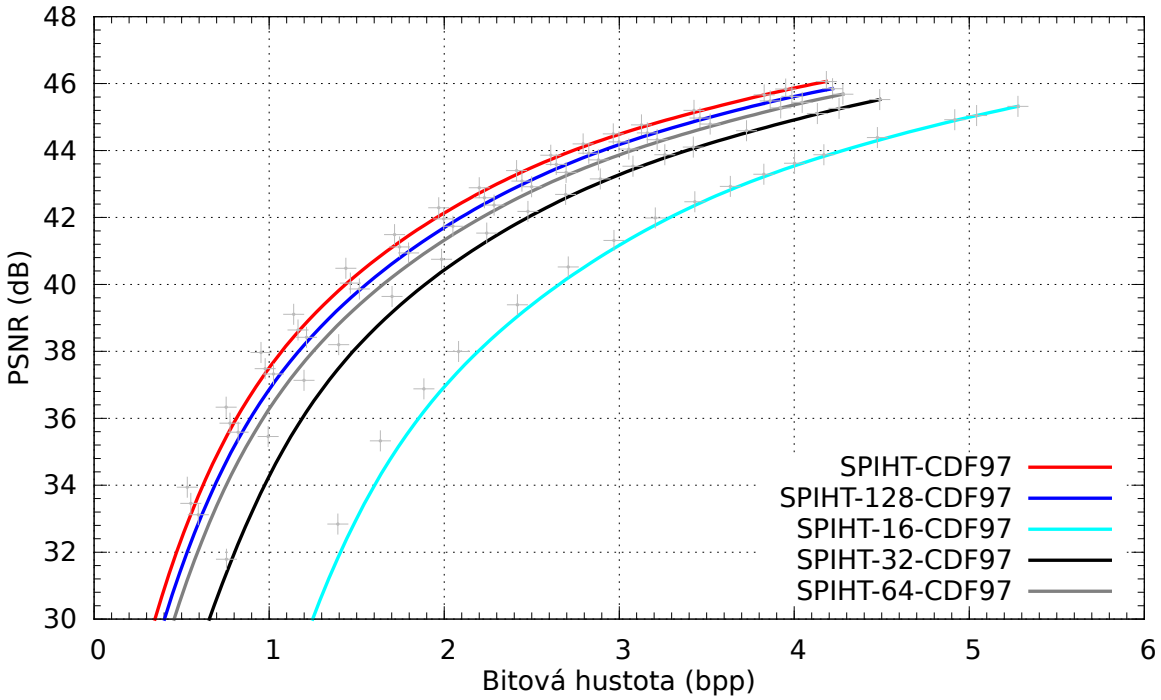
Obrázek 4.3: Graf vlivu vlnek na kvalitu komprese obrazu u algoritmu SPIHT. Zobrazuje průměr křivek. Nejlepších výsledků dosaduje vlnka CDF 9/7.

4.3 Dlaždice

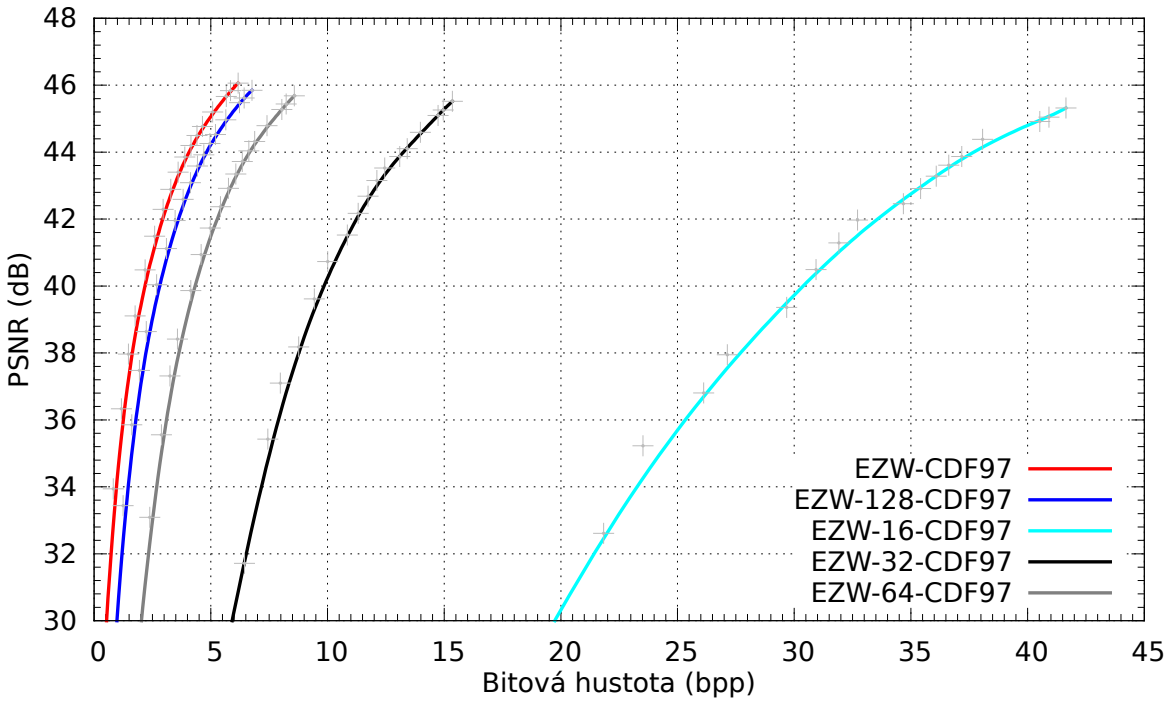
Jak můžeme vidět na grafu 4.4, dlaždice snižují kvalitu komprese. Ovšem použití dlaždic má smysl při menších kompresních poměrech, kde se neprojeví natolik nespojitost bloků.

U algoritmu EZW se použití dlaždic teoreticky vyplatí až od velikosti 64×64 pixelů, kde není kompresní poměr tak nevýhodný. Na obrázku 4.5 dokonce vidíme, že velikost komprimovaného obrazu u EZW s dlaždicí o velikosti 16×16 pixelů narostla průměrně přibližně na dvojnásobnou velikost (u 3 kanálového nekomprimovaného obrazu je bitová hustota 24bpp). Příloze B.5 je pak porovnán SPIHT a EBCOT s bloky.

Z grafu B.4 můžeme vyčíst, že vliv dlaždic na kvalitu komprese u obrazu „gradient1024“ (příloha A.0j) je výraznější právě z důvodu hladkých přechodů, kde nespojist bloků hraje větší roli.



Obrázek 4.4: Graf vlivu dlaždic na kvalitu komprese obrazu u algoritmu SPIHT a vlnky CDF 9/7. Zobrazuje průměr křivek. Vidíme, že se zmenšující se plochou dlaždice, se zhoršuje kompresní poměr i kvalita obrazu.

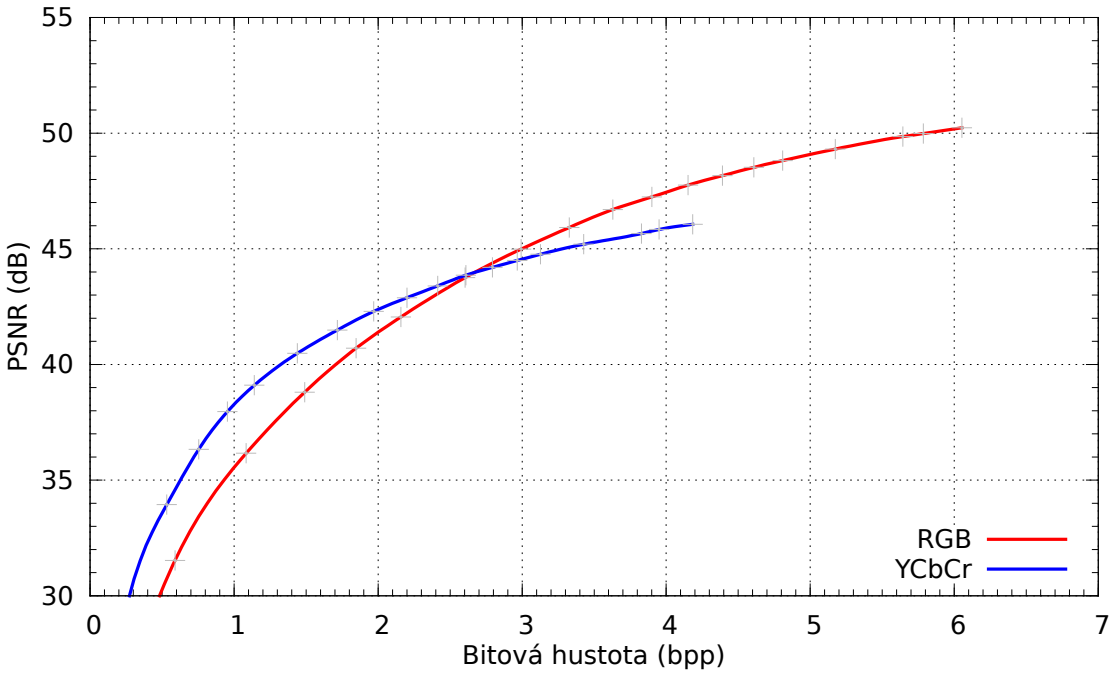


Obrázek 4.5: Graf vlivu dlaždic na kvalitu komprese obrazu u algoritmu EZW a vlnky CDF 9/7. Zobrazuje průměr křivek. EZW má zde u bloku 16×16 pixelů, při nižších poměrech komprese, téměř dvojnásobnou velikost komprimovaného souboru oproti původnímu obrazu (24bpp).

4.4 Barevný model

Zde byl pro porovnání použit algoritmus SPIHT s vlnkou CDF 9/7. Na grafu 4.6 vidíme, že se u RGB modelu při menším kompresním poměru dosahuje průměrně lepší kvality, ale za cenu horší komprese. Kdežto při použití $Y'C_bC_r$ lze dosáhnout lepších výsledků při vyšším kompresním poměru, např. graf B.6 nebo dříve zmíněný 3.3.

Ne vždy tomu tak je. Například, u obrázku „details1024“ (příloha A.0h) dosahuje barevný RGB model lepších výsledků, a až teprve při větším kompresním poměru se hodnoty začínají postupně dorovnávat (např. graf B.7.).



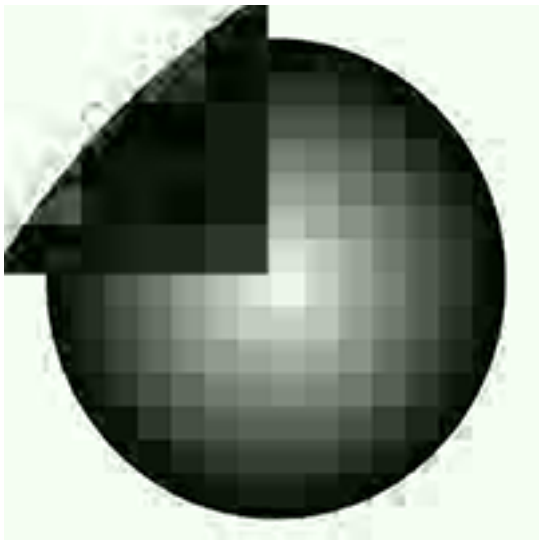
Obrázek 4.6: Graf vlivu barevného modelu na kvalitu komprese obrazu u algoritmu SPIHT a vlnky CDF 9/7. Zobrazuje průměr křivek. Zde vidíme, že RGB nedosahuje při větším kompresním poměru tak dobrých výsledků jako $Y'C_bC_r$.

4.5 Shrnutí testů

Nastavení parametrů komprese může mít velký vliv na její kvalitu a účinnost. Na její kvalitu má největší vliv použití dlaždic, kde např. u EZW, s nejmenšími dlaždicemi, se může velikost výsledného komprimovaného souboru navýšit na více jak dvojnásobek (obrázek 4.5).

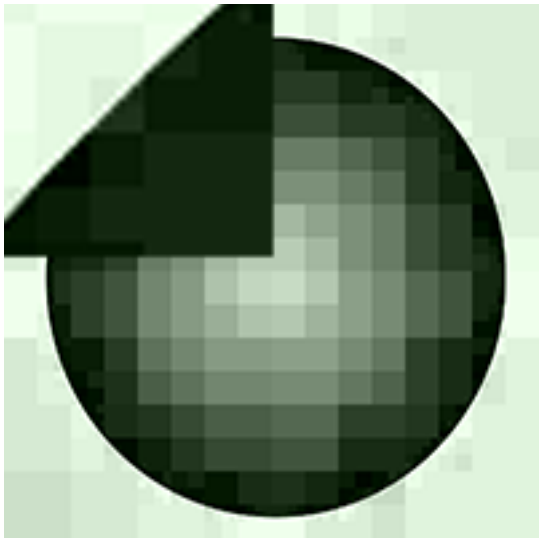
Výběr vlnek má také podíl na kvalitě. Vlnky s nižšími frekvencemi ztrácejí dříve detaily při rekonstrukci, než ty s větším počtem, např. vlnka Haar, která je sama nespojitá, trpí podobnými neduhy nespojitosti, jako při použití komprese s bloky (obrázek 4.6d).

(a) Kompresní poměr 1:540 u JPEG.



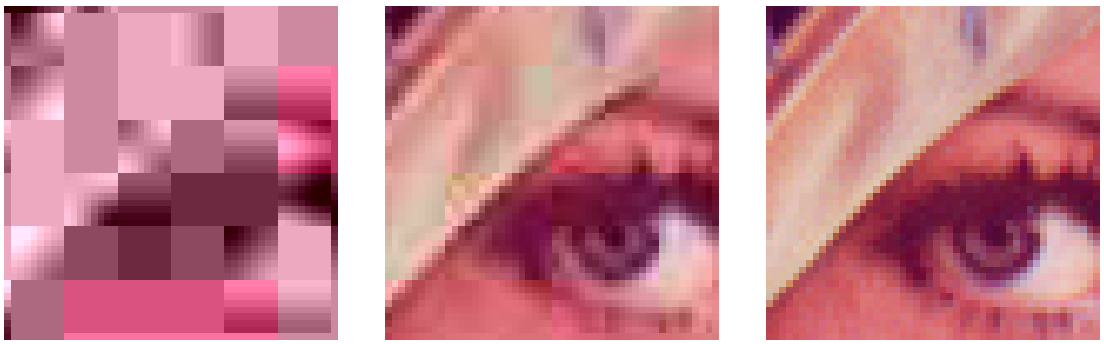
(b) Kompresní poměr 1:384 u SPIHT s 64×64 pixelů velkou dlaždicí s vlnkou CDF 9/7.

(c) Kompresní poměr 1:1536 u SPIHT bez dlaždic s vlnkou CDF 9/7.



(d) Kompresní poměr 1:323 u SPIHT bez dlaždic s vlnkou Haar.

Obrázek 4.6: Vliv dlaždic na kvalitu obrazu „ciclegradient1024“ (příloha A.0g). Vidíme zde výřezy, na kterých je zvětšen komprimovaný okraj. Také můžeme vidět, že výhoda těchto bloků může spočívat v tom, že se chyba rekonstrukce hrany nešíří mimo dlaždice. U vlnky Haar pozorujeme viditelné bloky i přesto, že dlaždice v tomto případě nebyly použity. Je to dáno tím, že vlnka Haar má nespojitý průběh.



(a) JPEG při PSNR 21,5dB (b) JPEG při PSNR 30,9dB (c) JPEG při PSNR 40,6dB

(d) SPIHT při PSNR 22,2dB (e) SPIHT při PSNR 30,5dB (f) SPIHT při PSNR 40,2dB

Obrázek 4.7: Porovnání JPEG a SPIHT s vlnkou CDF 9/7 při podobném PSNR. Vidíme zde, jak se s postupnou ztrátou kvality ztrací detaily z obrazu.

Kapitola 5

Závěr

Vlnková transformace má vysoký potenciál při zpracování obrazu, a to zejména v oblasti komprese. Tohoto potenciálu využívá poměrně nový standard JPEG2000, který v mnoha ohledech překonává nejen starší verzi tohoto standardu, JPEG, ale i jiné známé kompresní algoritmy, jako jsou EZW a SPIHT. Tyto algoritmy jsou překonány dokonce i první úrovní algoritmu EBCOT, tzv. Tier1.

Dále z práce vyplývá, že na účinnost komprese má velký vliv nejen použitá vlnka (průměrně nejlépe dopadla vlnka CDF 9/7), ale i velikost použité dlaždice, kde se zmenšující se velikostí její plochy klesá i účinnost samotné komprese.

Dalším zkoumaným vlivem na kvalitu komprese je barevný model $Y'C_bC_r$, který vykazuje lepší kvalitu obrazu při vyšších kompresních poměrech než barevný model RGB.

Navázat na tuto práci lze např. paralelizováním některých procesů výpočtu. Také by se dalo pokračovat úpravou stávajících algoritmů a dosáhnout tak ještě lepších výsledků kvality komprese.

Literatura

[1] JPEG 2000 Part I Final Committee Draft Version 1.0. 2000, ISO/IEC JTC 1/SC 29/WG 1 N1646R.

[2] A. Said, W. A. Pearlman: A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees. *IEEE Transactions on Circuits and Systems for Video Technology*, ročník 6, č. 3, 1996, ISSN 1051-8215.

[3] Addison Paul S.: *The Illustrated Wavelet Transform Handbook: Introductory Theory and Applications in Science, Engineering, Medicine and Finance*. Institute of Physics, 2002, ISBN 0-7503-0692-0.

[4] David Taubman: High Performance Scalable Image Compression with EBCOT. *IEEE Transactions on Signal Processing*, ročník 9, č. 7, 2000, ISSN 1057-7149.

[5] Jerome M. Shapiro: Embedded Image Coding Using Zerotrees of Wavelet Coefficients. *IEEE Transactions on Signal Processing*, ročník 41, č. 12, 1993, ISSN 1053-587X.

[6] Kamisetty Ramam Rao, Pat Yip: *The Transform and Data Compression Handbook*. CRC Press, 2001, ISBN 978-0849336928.

[7] Mallat S.: *A Wavelet Tour of Signal Processing, Third Edition*. Academic Press, 2009, ISBN 978-0-12-374370-1.

Příloha A

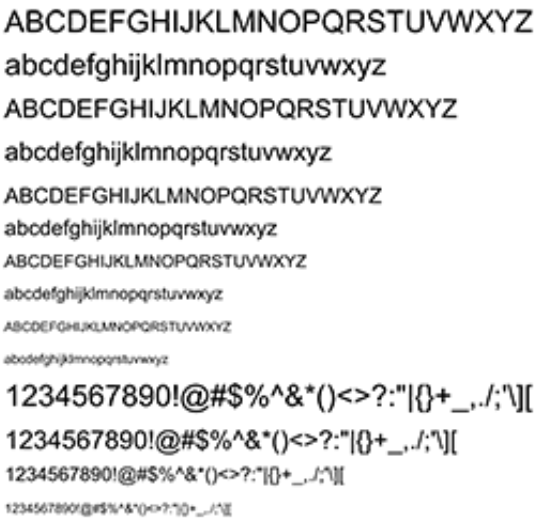
Použité obrazy v testech

(a) „beach1024“, rozměry 1024×1024 pixelů.

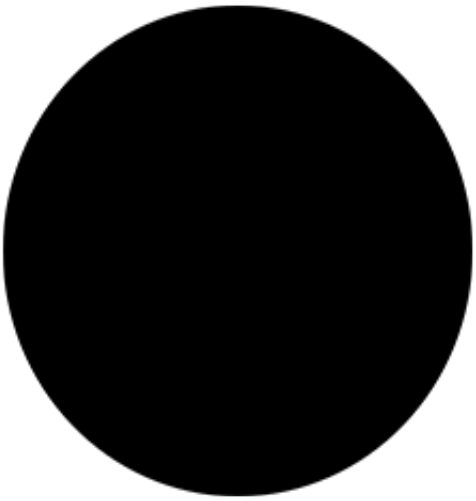
(b) „lena512“, rozměry 512×512 pixelů.

(c) „bird1920x1200“, rozměry 1920×1200 pixelů.

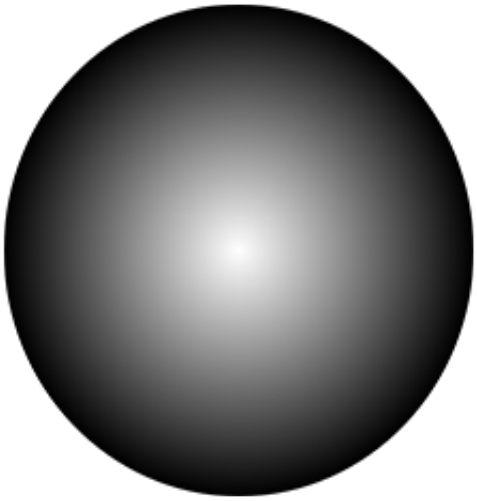
(d) „beach1920x1200“, rozměry 1920×1200 pixelů.



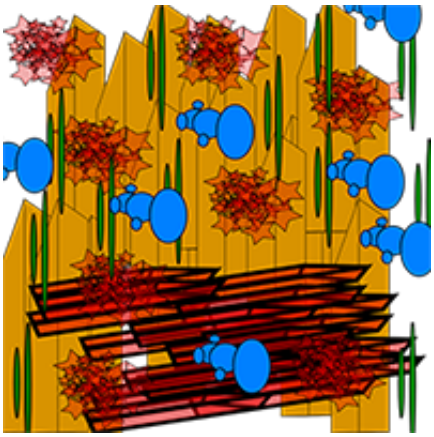
(e) „alphabet1024“, rozměry 1024×1024 pixelů.



(f) „cicle1024“, rozměry 1024×1024 pixelů.

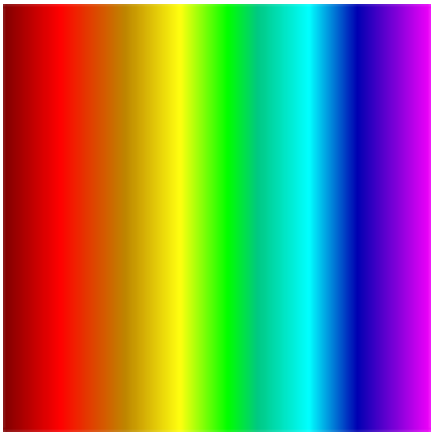


(g) „ciclegradient1024“, rozměry 1024×1024 pixelů.



(h) „details1024“, rozměry 1024×1024 pixelů.

(i) „geom1024“, rozměry 1024×1024 pixelů.



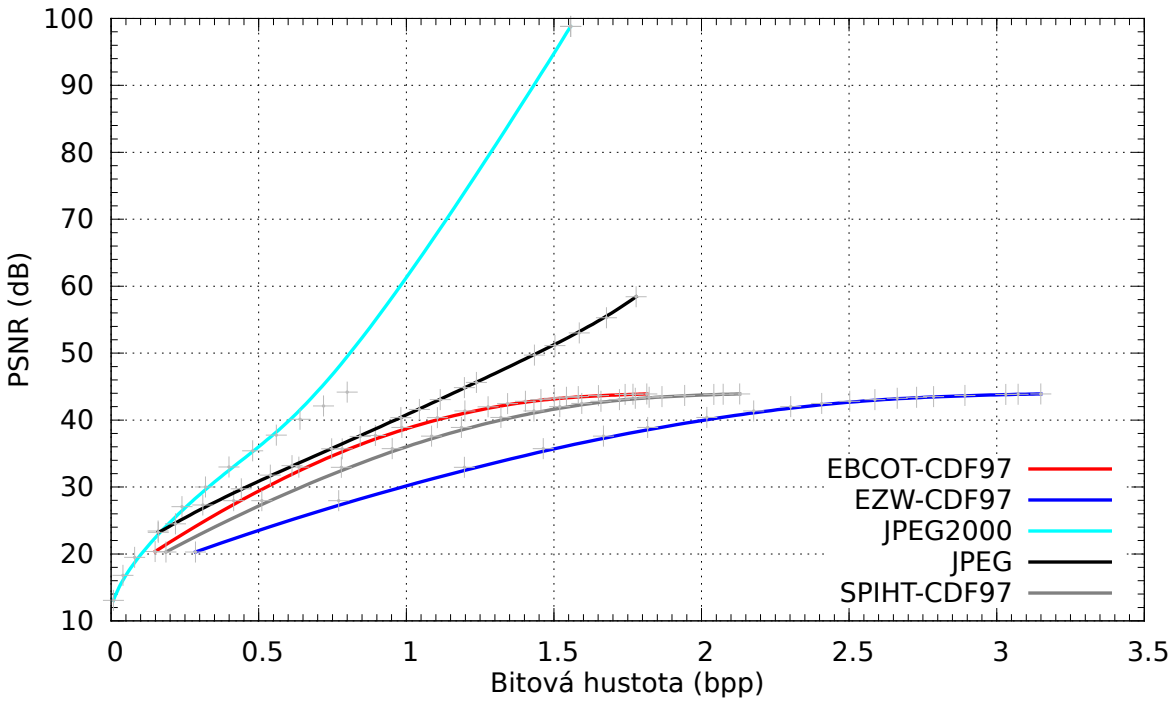
(j) „gradient1024“, rozměry 1024×1024 pixelů.

Obrázek A.0: Použité obrázky

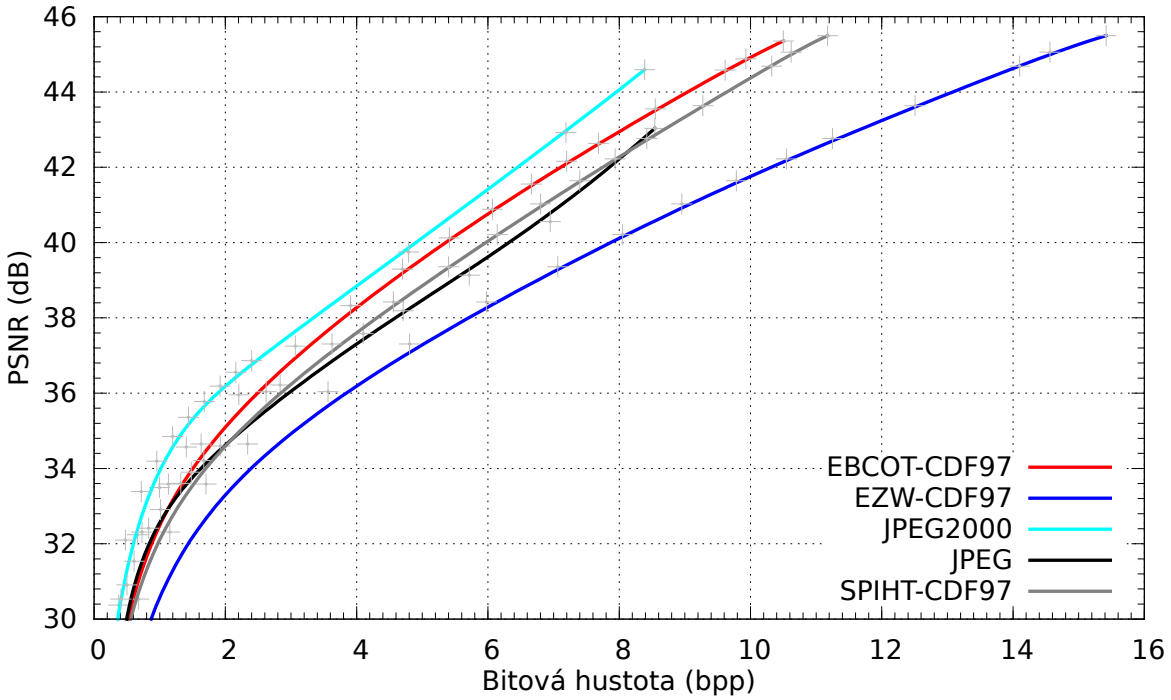
Příloha B

Vybrané grafy

B.1 Algoritmy

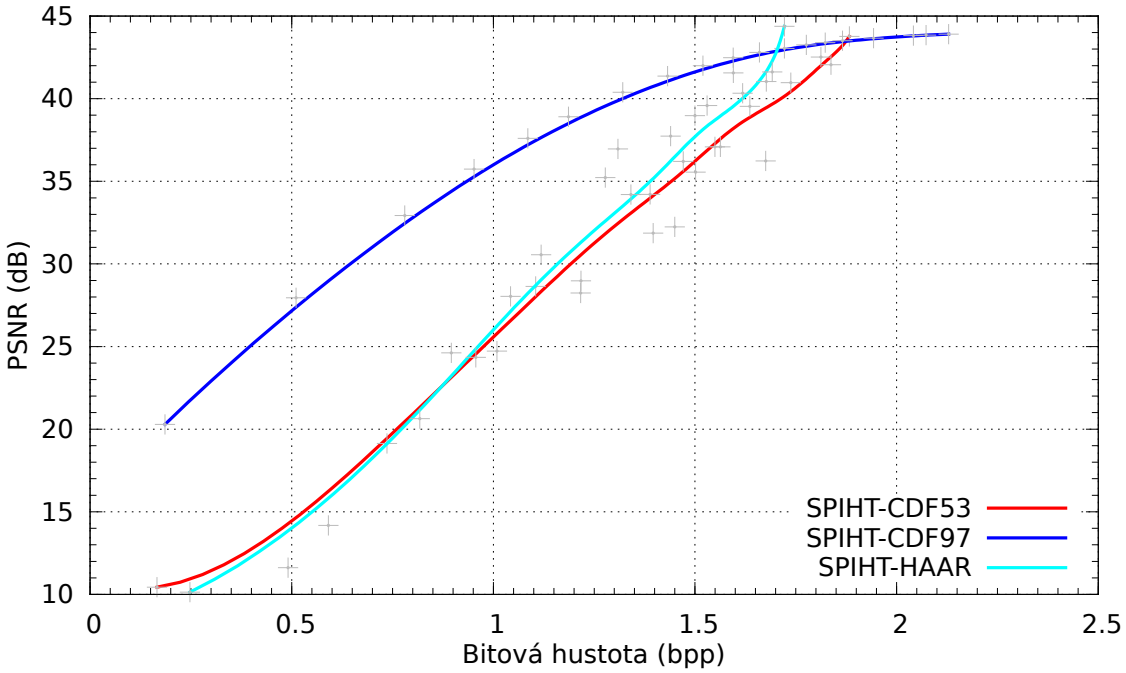


Obrázek B.1: Graf vlivu použitého algoritmu na kvalitu komprese obrazu „alphabet1024“ (příloha A.0e).



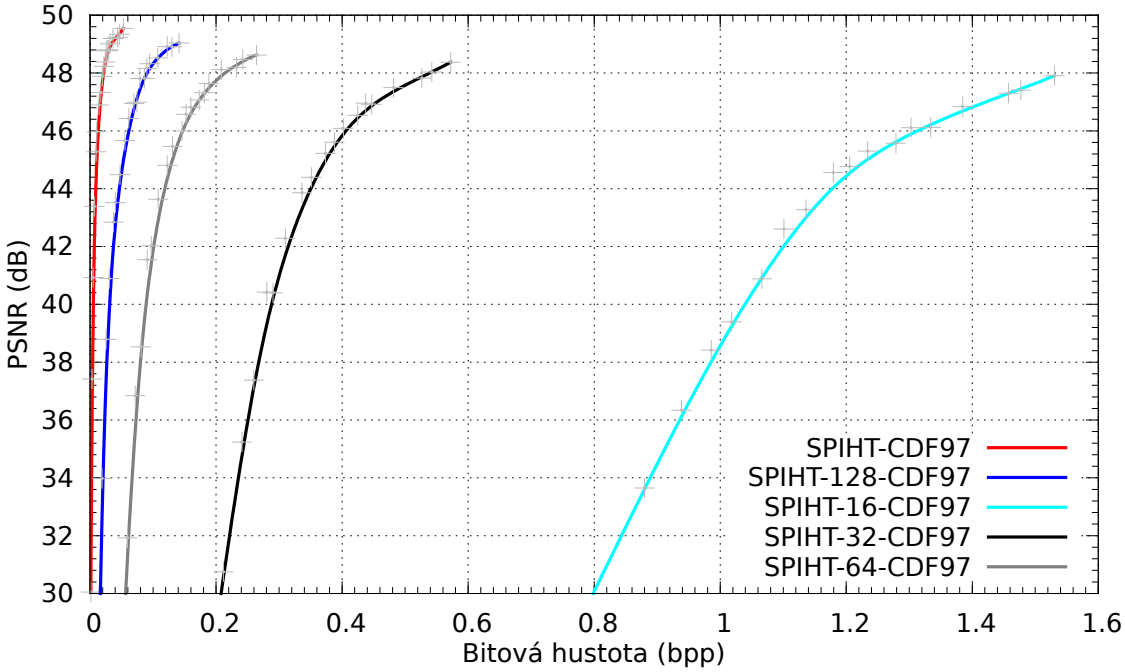
Obrázek B.2: Graf vlivu použitého algoritmu na kvalitu komprese obrazu „lena512“ (příloha A.1b). Velice podobné průběhy mají i zbylé realné obrazy.

B.2 Vlnky

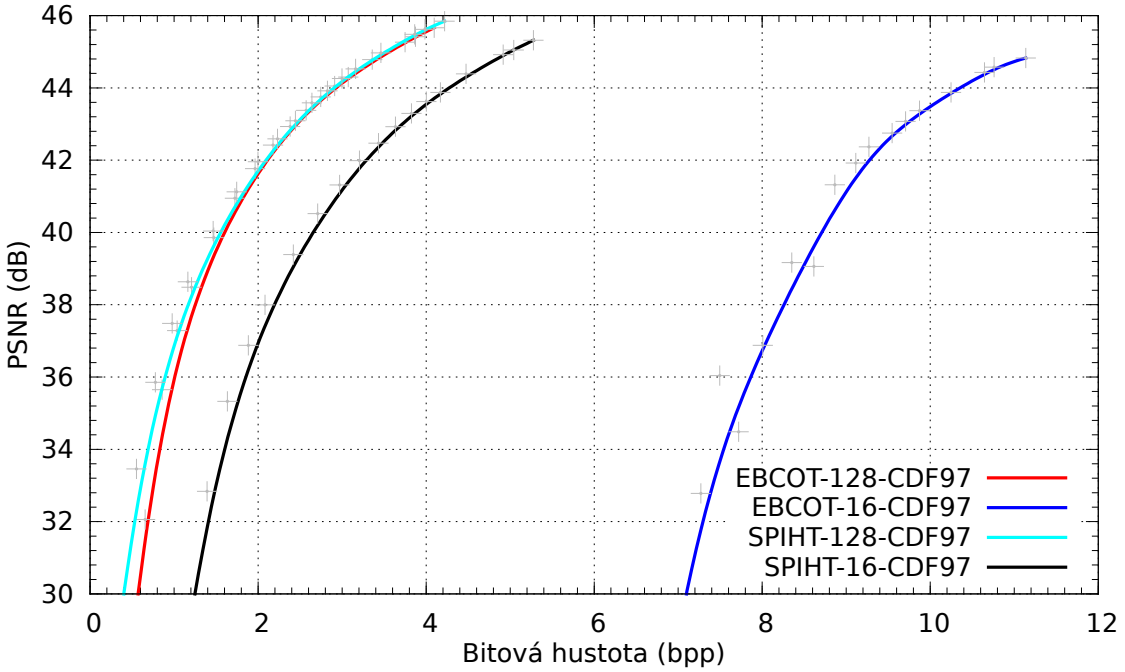


Obrázek B.3: Graf vlivu vlnek na kvalitu komprese obrazu „alphabet1024“ (příloha A.0e) u algoritmu SPIHT. Zde se z důvodu velkých jednobarevných ploch daří vlnce Haar více než v jiných připadech.

B.3 Dlaždice

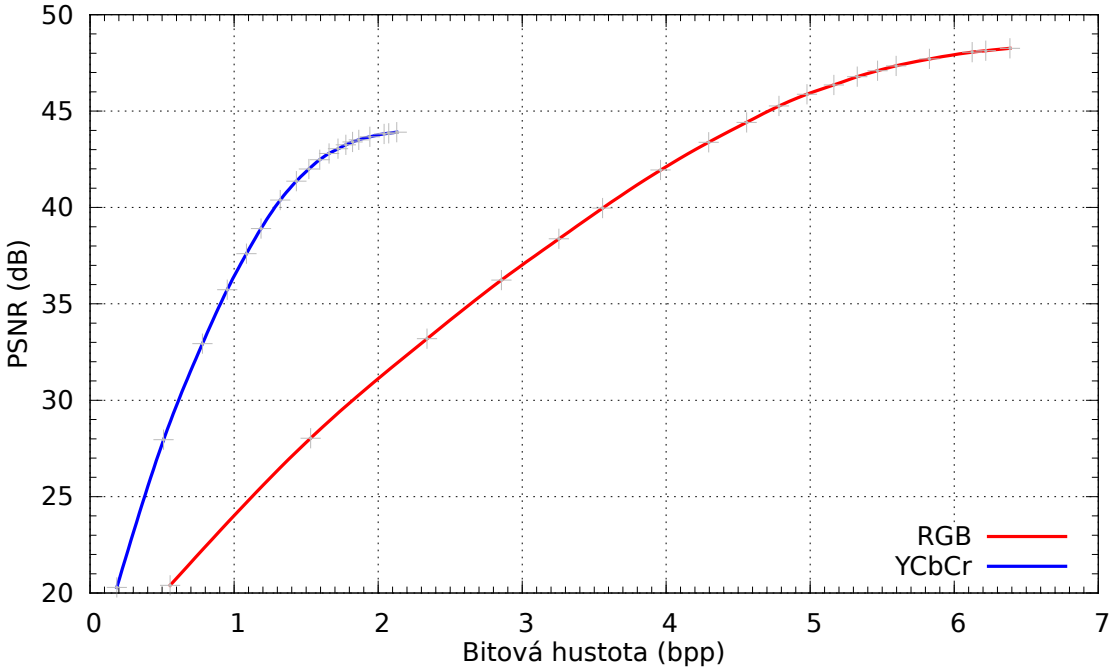


Obrázek B.4: Graf vlivu dlaždic na kvalitu komprese obrazu „gradient1024“ (příloha A.0j) u algoritmu SPIHT a vlnky CDF 9/7. Dlaždice zde hrají větší roli z důvodu hladkých přechodů.

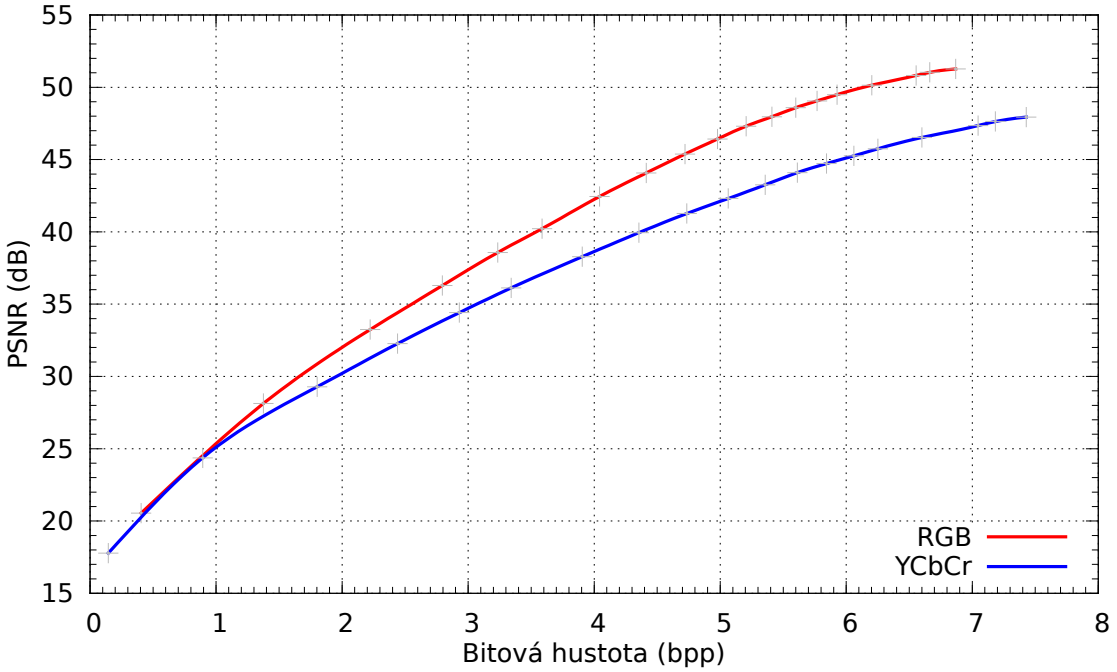


Obrázek B.5: Graf vlivu dlaždic na kvalitu komprese obrazu u algoritmu SPIHT a EBCOT s vlnkou CDF 9/7. Průměrné hodnoty křivek. I když je EBCOT průměrně lepší bez dlaždic, zde lehce ztrácí, obzvláště u menších bloků.

B.4 Barevný model



Obrázek B.6: Graf vlivu barevného modelu na kvalitu komprese obrazu „alphabet1024“ (příloha A.0e) u algoritmu SPIHT a vlnky CDF 9/7. Zde brarevný model dosahuje výrazně lepších výsledků.



Obrázek B.7: Graf vlivu barevného modelu na kvalitu komprese obrazu „details1024-rgb“ u algoritmu SPIHT a vlnky CDF 9/7. U toho obrazu naopak RGB model dosahuje lepších výsledků. Je to zřejmě tím, že obsahuje mnoho různých barev, které dobře zapadají do RGB modelu (stejně dobře dopadl i u obrazu „gradient1024-rgb“ z přílohy A.0j)).